



**Calhoun: The NPS Institutional Archive**  
**DSpace Repository**

---

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

---

1990-09

# Design and implementation of an Intelligent Cost Estimation Model for Decision Support System software

Blazer, William E.

Monterey, California: Naval Postgraduate School

---

<http://hdl.handle.net/10945/34873>

---

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

*Downloaded from NPS Archive: Calhoun*



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

**Dudley Knox Library / Naval Postgraduate School**  
**411 Dyer Road / 1 University Circle**  
**Monterey, California USA 93943**

<http://www.nps.edu/library>

AD-A241 322



2

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



DTIC  
ELECTE  
OCT 10 1991  
S B D

### THESIS

DESIGN AND IMPLEMENTATION OF AN INTELLIGENT  
COST ESTIMATION MODEL FOR DECISION  
SUPPORT SYSTEM SOFTWARE

by

William E. Blazer

September 1990

Thesis Advisor:

Tung X. Bui

Approved for public release; distribution is unlimited

91-12679



91 10 7 015

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b. OFFICE SYMBOL (If applicable) Code 37		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000			7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification) DESIGN AND IMPLEMENTATION OF AN INTELLIGENT COST ESTIMATION MODEL FOR DECISION SUPPORT SYSTEM SOFTWARE					
12. PERSONAL AUTHOR(S) Blazer, William E.					
13a. TYPE OF REPORT Master's Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) 1990, September	
15. PAGE COUNT 126					
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Software Cost Estimation Model; COCOMO; Function Point		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Recently, Decision Support Systems (DSS) have increased in importance and usage. However, these increases have not carried over into developing better models to estimate the real cost of developing the DSS. This thesis explores various estimation methods that seem pertinent to DSS. It advocates the use of a combination of modeling tools particularly tailored to the users' environment. An Intelligent Cost Estimation Model (ICEM) for Decision Support System software is proposed. To promote user-friendliness, ICEM uses a rule-based front-end interface coupled to a spreadsheet program. For comparison purposes the current version of ICEM includes the Intermediate COCOMO model, the Intermediate COCOMO model particularly calibrated for the in-house DSS development environment, and a parametric model which incorporates the function point size metric.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. Tung X. Bui			22b. TELEPHONE (Include Area Code) (408) 646-2630		22c. OFFICE SYMBOL Code AS/Bd

Approved for public release; distribution is unlimited

Design and Implementation of an Intelligent Cost Estimation  
Model for Decision Support System Software

by

William E. Blazer  
Lieutenant, United States Navy  
B.S., Virginia Military Institute, 1984

Submitted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS


from the

NAVAL POSTGRADUATE SCHOOL  
September 1990

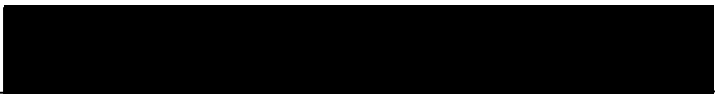
Author:

  
William E. Blazer

Approved by:

  
Tung X. Bui, Thesis Advisor

  
Tarek Abdel-Hamid, Second Reader

  
David R. Whipple, Chairman,  
Department of Administrative Sciences

## ABSTRACT

Recently, Decision Support Systems (DSS) have increased in importance and usage. However, these increases have not carried over into developing better models to estimate the real cost of developing the DSS. This thesis explores various estimation methods that seem pertinent to DSS. It advocates the use of a combination of modeling tools particularly tailored to the users' environment. An Intelligent Cost Estimation Model (ICEM) for Decision Support System software is proposed. To promote user-friendliness, ICEM uses a rule-based front-end interface coupled to a spreadsheet program. For comparison purposes the current version of ICEM includes the Intermediate COCOMO model, the Intermediate COCOMO model particularly calibrated for the in-house DSS development environment, and a parametric model which incorporates the function point size metric.



<b>Accession For</b>	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

### THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user. Additionally, the ICEM program is available from the thesis advisor, however, since VP-Expert and VP-Planner Plus are a copyright of Paperback Software International, they can not be included for distribution.

## TABLE OF CONTENTS

I.	INTRODUCTION -----	1
	A. BACKGROUND -----	1
	B. SCOPE -----	1
	C. METHODOLOGY -----	2
	D. ORGANIZATION -----	3
II.	A FRAMEWORK TO CLASSIFY COST ESTIMATION MODELS -----	4
	A. BASILI'S CLASSIFICATION SCHEME -----	5
	B. THIBODEAU'S CLASSIFICATION SCHEME -----	7
	C. SUMMARY -----	9
III.	SIZE METRICS -----	11
	A. SOURCE LINES OF CODE -----	11
	B. FUNCTION COUNTS -----	12
	C. HALSTEAD'S SOFTWARE SCIENCE -----	13
	D. FUNCTION POINTS -----	15
	E. OTHER PRODUCT ATTRIBUTES -----	17
IV.	EXISTING COST ESTIMATION MODELS -----	19
	A. BACKGROUND -----	19
	B. COST ESTIMATION MODELS -----	20
	C. SUMMARY -----	27
V.	COCOMO MODEL -----	29
	A. OVERVIEW -----	29
	B. BASIC MODEL -----	29

C.	INTERMEDIATE COCOMO -----	33
D.	CALIBRATING THE COCOMO MODEL -----	36
VI.	FUNCTION POINT ANALYSIS MODEL -----	39
A.	BACKGROUND -----	39
B.	CALCULATING FUNCTION POINTS -----	40
C.	STEPS (1 & 2) COUNT AND CLASSIFY FUNCTION TYPES -----	40
D.	(STEP 3) ADJUST FOR PROCESSING COMPLEXITY --	48
E.	(STEP 4) MAKE THE FUNCTION POINTS CALCULATION -----	49
F.	SUMMARY -----	51
VII.	INTELLIGENT COST ESTIMATION MODEL (ICEM) -----	52
A.	INTRODUCTION -----	52
B.	OVERALL ARCHITECTURE OF ICEM -----	54
VIII.	USING ICEM FOR EFFORT ESTIMATION OF DSS SOFTWARE -----	59
A.	INTRODUCTION -----	59
B.	THE DSS DATABASE -----	59
C.	CALIBRATING THE COCOMO MODEL FOR DSS -----	63
D.	DISCUSSIONS -----	65
IX.	CONCLUSIONS AND RECOMMENDATIONS -----	67
A.	CONCLUSIONS -----	67
B.	RECOMMENDATIONS FOR FURTHER RESEARCH -----	68
APPENDIX A:	SAMPLE SESSION -----	69
APPENDIX B:	SOURCE CODE -----	80
	LIST OF REFERENCES -----	115
	INITIAL DISTRIBUTION LIST -----	119



## I. INTRODUCTION

### A. BACKGROUND

Recently, Decision Support Systems (DSS) have increased in importance and usage. Likewise, there has been an increase in literature regarding the design and development of DSS software. Unfortunately, these increases have not carried over into developing better models to estimate the real cost of developing the DSS. Current literature argues that a DSS should be developed quickly and inexpensively; however, user requirements in terms of interface and modeling capabilities often significantly increase development cost [Ref. 1].

As the cost of software projects has soared over the past decade there have been several attempts to develop cost estimation models that would allow a manager to accurately predict the development cost of software projects. Some of these models are addressed in Chapter IV. With cost and demand for new software at an all-time high and increasing backlogs of undeveloped software projects, the development of better cost estimation models is more important today than it ever has been.

### B. SCOPE

The purpose of this thesis is to explore the possibility of combining the advantages of parametric and heuristic

modeling to better estimate the development cost of DSS software.

The main thrust of this study will be in the design and implementation of an Intelligent Cost Estimation Model (ICEM) that would allow a DSS builder to predict and refine development cost. Due to the considerable difficulty of DSS software estimation, this work will focus on small scale DSS applications developed by small programming groups whose development tools consist of high level programming languages. Furthermore, it will concentrate on developing a calibrated model for the DSS environment based on the COCOMO and function point concepts.

#### C. METHODOLOGY

When estimating the cost of software development the quality of the estimation depends upon the precision of the software specification. By their nature, Decision Support Systems deal with ill-defined problems. This, in turn, makes it difficult to accurately estimate the cost of developing the DSS. Two research methodologies can be applied in developing cost estimation models. These include the parametric and heuristic approaches. This research attempts to combine these two approaches to gain some insights on the DSS cost behavior. An integrated design approach will be used to implement the model. It will consist of a rule-based expert system which is coupled to a spreadsheet model. The expert system is used to

gather heuristic information for cost estimation, while the spreadsheet provides the framework for the parametric model which performs statistical calculations.

#### D. ORGANIZATION

This thesis is broken down into nine chapters with two appendices. After the introduction in Chapter I, the thesis begins in Chapter II by first looking at the framework in which cost estimation models are classified. The issue of size metrics and how they effect the development of cost estimation models are addressed in Chapter III. Different methods used in measuring software size are also addressed in this chapter along with the advantages and disadvantages of each metric. Chapter IV presents an overall review of several cost estimation models currently available. The COCOMO and Function Point models are studied in greater detail in Chapters V and VI respectively. The ICEM model is presented in Chapter VII. Chapter VIII provides an empirical analysis of the ICEM model based on collected data. Due to the inaccurate nature of some required information, calibration was performed only for the COCOMO model. Chapter IX contains the conclusion and future recommendations. A sample session with the ICEM model showing screen formats is provided in Appendix A. Finally, the source code for the ICEM model is provided in Appendix B.

## II. A FRAMEWORK TO CLASSIFY COST ESTIMATION MODELS

There have been many models developed to estimate the cost of software development. Most of these models are empirical models that use information gathered from previous projects to make future predictions of current projects such as time, effort, and cost requirements. They derive basic equations from past projects using statistical tools such as linear regression. Models typically utilize one or a combination of five major parameters for estimation: productivity, schedule, cost, quality, and size. The most common parameter used for estimation is size (see Chapter III).

Models have been classified under many different formats. Many models are defined as being either micro or macro models. A micro model derives effort estimations from small pieces of information which have been scaled upwards (bottom-up approach). The total effort is derived from the sum of all of the smaller effort estimates. On the other hand, a macro model is based upon a view of the big picture (top-down approach). Effort is estimated for the entire product development and then proportioned between the separate development activities.

## A. BASILI'S CLASSIFICATION SCHEME

One of the disadvantages in using the micro/macro scheme for classifying models is that it is very limited in scope; it only separates models into two large groups. A more detailed scheme was developed by Victor Basili, who distinguished models according to the type of formula they used to calculate total effort [Ref. 2]. He defined models as being single-variable if only one basic variable was used as a predictor of effort and multi-variable if several were used. He further defined models as being either static single-variable, static multivariable, or dynamic multivariable.

### 1. Static Single-variable Model

In a static single-variable model, a unique variable such as SLOC (Source Lines of Code) is utilized in the derived equations to make predictions about other variables such as cost or time. The basic effort equation of this model takes on the following form:

$$\text{Effort} = a * \text{Size}^b$$

where  $a$  and  $b$  are estimated constants derived through linear regression of a historical database. The Walston-Felix and the basic COCOMO cost estimation models are examples of single-variable models. A description of all cost estimation models referred to in this chapter may be found in Chapter IV

with the exception of the COCOMO and Function Point models which are described in Chapters V and VI respectively.

## 2. Static Multivariable Models

Cost estimating models, as defined by Basili, may also be categorized as static multivariable. A model is considered multivariable if it is based on several parameters and static if a single effort value is calculated by the model formula. The static multivariable models use the additional parameters to make adjustments to the original estimation. These adjustments are most often based on historical data. This model is further subcategorized as being either an adjusted baseline model, an adjusted table-driven model, or a multi-parameter equation model.

### a. Adjusted Baseline Model

The adjusted baseline model uses a single-variable baseline equation which is adjusted in some way by a set of other variables. The Intermediate COCOMO model fits this category as its baseline effort estimate relies only upon project size and it applies a set of adjustment multipliers to its effort equation.

### b. Adjusted Table-Driven Model

The adjusted table-driven model uses a baseline estimate adjusted by a set of variables whose relationships are defined in tables built from historical data. An example of this is the Wolverton model. This model contains a basic algorithm which involves categorizing the software routines.

### c. Multi-parameter Model

In the multi-parameter model, a base formula is used which contains several variables. The GRC model contains several multiparameter equations containing more than one variable.

### 3. Dynamic Multivariable Model

The final category defined by Basili is the dynamic multivariable model. Unlike the static model, the dynamic model does not have a single basic variable. Rather, it contains a set of inter-dependent variables which respond to changes over a period of time, such as staffing levels. The Putnam model is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project.

Figure 2.1 shows the model categories and their relationships as described by Basili.

## B. THIBODEAU'S CLASSIFICATION SCHEME

Another categorization scheme for cost estimation models was developed by Robert Thibodeau. Thibodeau grouped models into three separate categories based on the method the model uses in making estimations. These include: regression, heuristic, and phenomenological. [Ref. 4]

### 1. Regression Model

In the regression model, parameters are developed for a single cost estimation equation by using linear regression

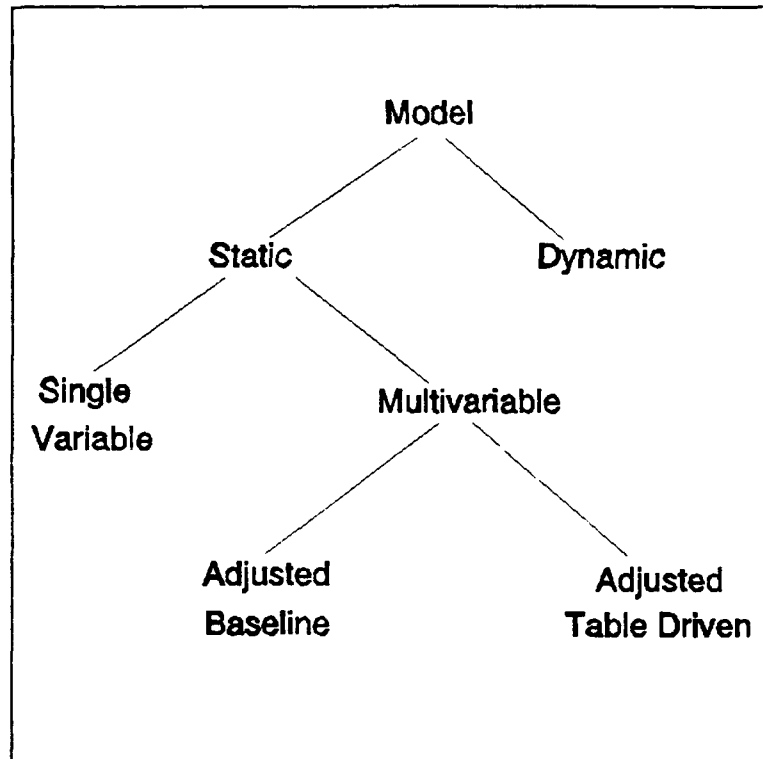


Figure 2.1 Model Relationships  
[Ref. 3:p. 32]

on available data. This model is specific to the environment in which it is developed. Examples of regression models include the Doty, COCOMO, and GRC models.

## 2. Heuristic Model

Unlike the regression model, the heuristic model is considered to be free from any singular mathematical formulation. Heuristic models usually combine a number of different estimating techniques. They provide a flexible approach which utilizes observations of relations, interpretations, and a certain amount of subjectivity. They provide an intuitive development of an estimate which takes advantage of



previous estimates and adjustment factors. Examples of heuristic models include the Boeing, Price S, and Wolverton models.

### 3. Phenomenological Model

The third type of model described by Thibodeau is the phenomenological model. This category was developed for and is based on the SLIM model. It is unique in the fact that it is the only model to use observed basic relationships which are unrelated to the software development process. These relationships are more representative of scientific law rather than adaptive interrelationships which are used in heuristic models.

### C. SUMMARY

From all of the previous category listings it is evident that there are many methods for classifying cost estimation models. These methods are often useful when analyzing different cost estimation models. However, regardless of the classification of the model, almost all of the cost estimation models underestimate their values. Additionally, if the model is not properly calibrated to the user's environment, this estimation error may be as great as 500 percent or more [Refs. 5-8]. With this in mind, it is the intent of this thesis to develop an integrated automated model (ICEM) which utilizes two of the best models currently available. The ICEM model will combine the benefits of a parametric model (COCOMO) with

the advantages of a heuristic model (Function Point) in order to provide a better cost estimating tool. By using models based on different classification schemes, the user will better be able to validate estimations. Additionally, this model will provide the user with two alternative methods for making cost estimations. This allows the user the ability to select the model most appropriate for the given situation. Since the ICEM model will be calibrated to the unique environment of Decision Support Systems, it is only recommended for this environment.

### III. SIZE METRICS

Almost all of the cost estimation models currently available use some form of size measurement or size metric for making productivity and effort estimates. Likewise, most of the error in estimating cost of software development is attributed to error in estimating the size of the program. An accurate and consistent method for obtaining size measurements is essential to the success of most cost estimation models. As DeMarco [Ref. 9:p. 3] states in reference to metrics, "You can't control what you can't measure." This is also true for estimation, you can't estimate without proper input measurements.

#### A. SOURCE LINES OF CODE

One of the most common methods for measuring the size of a project is by measuring the number of source lines of code (SLOC) which it contains. It would seem that measuring the number of lines of code would be an easy and accurate process for providing a consistent size measurement; however, in order to do this one must first determine what constitutes a line of code. This can vary depending on the desired output. For instance, when trying to determine the functional size of a program it is generally accepted that only executable statements are counted as lines of code; therefore, comments,

blank lines, and data are not considered. But, if determining the total amount of effort is the desired result then all factors should be included in the measurement.

Furthermore, there still remains a problem that the effort required for writing a line of executable code may be easy or difficult depending on the task. Therefore the size of a program measured in SLOC does not provide a standard basis for determining the effort involved. For example, a 100-line program could take as little as a day to develop or as much as a week depending on the complexity of the program task.

Another problem with using SLOC measurements is that programs are written in many different languages, such as Pascal, ADA, Fortran, Cobol, and Assembly language. The amount of code required to perform a task in one language does not necessarily equate to the number of lines that another programming language would require to perform the same task. Additionally, SLOC unfairly penalizes fourth generation languages for their additional complexity. This point is demonstrated in the empirical analysis of Chapter VIII. Since most DSS are developed using fourth generation languages this is very disadvantageous.

#### B. FUNCTION COUNTS

In order to overcome the problems associated with SLOC, there have been many other methods introduced for measuring program size. Function counts is one such method which

concentrates on measuring the amount of functions in a program. Conte et al. [Ref. 10:p. 43] defined a function:

...as a collection of executable statements that performs a certain task, together with declarations of the formal parameters and local variables manipulated by those statements.

The advantage in using the number of functions as a size metric is that essentially, the number of functions will remain the same regardless of the language the program is written in. Additionally, during early life cycle development when the code has not yet been developed, it is often easier to estimate the number of functions that will be required than to estimate the SLOC. The down side to this method is that there is considerable overhead and cost in counting the number of functions. This overhead discourages the definition of a small function. Theoretically, a function could be as small as a single statement or as large as an entire procedure depending on how it is defined. This could lead to consistency problems and as programs are separated into larger functions the benefits of this method are lost.

#### C. HALSTEAD'S SOFTWARE SCIENCE

As previously discussed, one of the major problems with SLOC is consistency, since some lines of code are more difficult to code than others. One solution to this problem is to give more weight to lines that are more complex. Maurice Halstead developed such a scheme in his metric called

Software Science [Ref. 11]. Halstead's method uses measurements of operators and operands.

Operators are primarily symbols or keywords that specify an action. They consist of arithmetic symbols (such as +, -, and /), command names (such as IF, IF..THEN..ELSE, or DO..WHILE), special symbols (such as :=, braces, and semicolons), and finally grouping functions (such as BEGIN..END). Since BEGIN..END performs the same function it is considered one operator. Operands are the symbols used to represent data. They consist of variables, constants, and labels.

Software Science uses measurements of operators and operands to make predictions about a program's length, volume, difficulty, level, and effort required. The length of a program is calculated as a dimensionless quantity but can be converted to SLOC by dividing by a constant which is language-dependent [Ref. 10:p.41]. The volume of a program is also a size measurement, but it is in terms of the minimum number of bits required for programming. Volume is dependent upon a measurement that is referred to as a program's vocabulary.

The basic metrics of Software Science are defined as:

$n_1$  = number of distinct operators in a program

$n_2$  = number of distinct operands in a program

$N_1$  = number of occurrences of operators in a program

$N_2$  = number of occurrences of operands in a program

Estimations are accomplished by using the following equations:

$N$	= Observed Program Length	$= N_1 + N_2$
$\hat{N}$	= Estimated Program Length	$= n_1(\log_2(n_1)) + n_2(\log_2(n_2))$
$n$	= Program Vocabulary	$= n_1 + n_2$
$V$	= Program Volume	$= N(\log_2(n))$
$D$	= Program Difficulty	$= (n_1/2)/(N_2/n_2)$
$L$	= Program Level	$= 1/D$
$E$	= Effort	$= V/L$

It has not been sufficiently proven that the metrics proposed by Halstead are actually any better at estimating size than SLOC and there have been many people who have questioned its effectiveness [Refs. 10; 12-15]. In view of this, and with the increased cost and overhead associated with the Halstead and other metrics, SLOC has continued to be the dominant size metric. Additional information regarding the Halstead and many other software metrics and models may be found in a comprehensive collection of related articles by Victor Basili [Ref. 16].

#### D. FUNCTION POINTS

Probably one of the more successful and rapidly growing methods for measuring size has been Albrecht's Function Point Analysis method [Refs. 17; 18]. The function point metric was also developed as an alternative to LOC as the principle sizing metric. Function Point metrics do not measure LOC,

rather, they focus on program functionality. This is not to be confused with the function count method previously discussed. Function Point Analysis is not simply a measurement of the number of functions. Rather, it measures five specific areas which include: inputs, outputs, interfaces, files and inquiries. A complexity factor is used to adjust the numeric values of each of the five areas. These values are then summed to obtain a function point count which can be used as a dimensionless sizing metric or further equated to a SLOC measurement.

Most of the cost estimation models that have been developed are unable to make accurate estimates early in the development phase of a project. This is because most of the models rely on SLOC as the primary size metric and estimating SLOC early in the life cycle is very difficult and highly inaccurate. Boehm [Ref. 19:p. 311] illustrated the difficulty cost models have in making accurate estimates early in the life cycle of a project. This is shown in Figure 3.1.

The method of calculating function points is described in detail in Chapter VI. One of the advantages of using function points is that they can be computed early in the development cycle, essentially after the requirements and functional specifications are written. Additionally, by concentrating on program functionality the problems associated with using different languages disappear. However, some researchers also point out some non-negligible problems associated with this



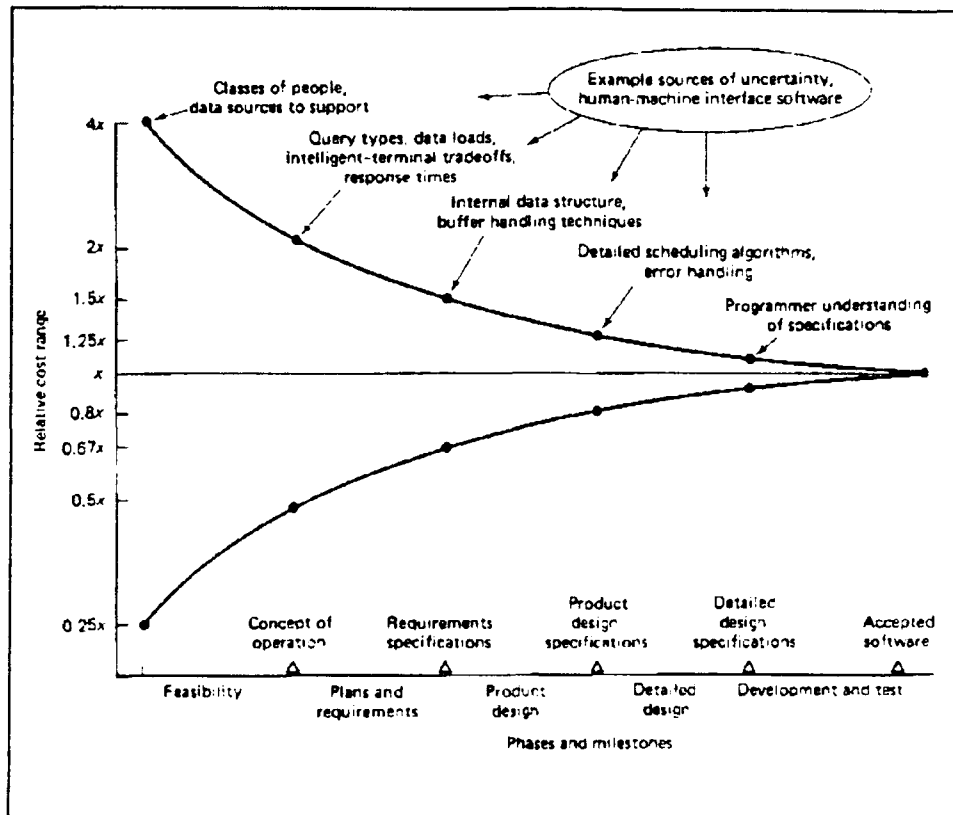


Figure 3.1 Software Cost Estimation Accuracy Versus Phase [Ref. 19:p. 311]

method. First, the counting of the function points is prone to subjective assessment. Second, it is difficult to collect measurements on a specific information domain after-the-fact. Finally, since the function point calculation is a dimensionless quantity, it might convey little meaning. [Ref. 20:p. 94]

#### E. OTHER PRODUCT ATTRIBUTES

Although size estimates are very important in developing a cost estimation model there are several other major

attributes which are also utilized. Boehm [Ref. 21:p. 11] identified five major attributes used in cost estimation and the factors that measure them. Figure 3.2 displays these attributes and factors in relation to their use by the various models described in Chapter IV.

GROUP	FACTOR	SDC 1966	TRW 1972	PUTNAM SLIM	DOTY	RCA PRICES	IBM	BOEING 1977	GRC 1979	COCOMO	SOFCOST	DSN	JENSEN
SIZE ATTRIBUTES	SOURCE INSTRUCTIONS			X	X		X	X		X	X	X	X
	OBJECT INSTRUCTIONS	X	X		X	X							
	NUMBER OF ROUTINES	X				X					X		
	NUMBER OF DATA ITEMS						X			X	X		
	NUMBER OF OUTPUT FORMATS								X			X	
	DOCUMENTATION				X		X				X		X
PROGRAM ATTRIBUTES	NUMBER OF PERSONNEL			X		X	X	X			X		X
	TYPE	X	X	X	X	X	X	X			X		
	COMPLEXITY		X	X		X	X			X		X	X
	LANGUAGE	X		X				X	X		X	X	
	REUSE			X		X		X	X		X	X	X
	REQUIRED RELIABILITY			X		X				X	X	X	X
COMPUTER ATTRIBUTES	DISPLAY REQUIREMENTS				X						X		X
	TIME CONSTRAINT		X	X	X	X	X	X	X	X	X	X	X
	STORAGE CONSTRAINT			X	X	X	X			X	X	X	X
	HARDWARE CONFIGURATION	X				X							
	CONCURRENT HARDWARE DEVELOPMENT	X			X	X	X			X	X	X	X
	INTERFACING EQUIPMENT SW										X	X	
PERSONNEL ATTRIBUTES	PERSONNEL CAPABILITY			X		X	X			X	X	X	X
	PERSONNEL CONTINUITY						X					X	
	HARDWARE EXPERIENCE	X		X	X	X	X		X	X	X	X	X
	APPLICATIONS EXPERIENCE		X	X		X	X	X	X	X	X	X	X
	LANGUAGE EXPERIENCE			X		X	X		X	X	X	X	X
						X	X			X	X	X	X
PROJECT ATTRIBUTES	TOOLS AND TECHNIQUES			X		X	X	X		X	X	X	X
	CUSTOMER INTERFACE	X					X				X	X	
	REQUIREMENTS DEFINITION	X			X		X				X	X	X
	REQUIREMENTS VOLATILITY	X			X	X	X		X	X	X	X	X
	SCHEDULE			X		X				X	X	X	
	SECURITY						X				X	X	
	COMPUTER ACCESS			X	X		X	X		X	X	X	X
	TRAVEL/REHOSTING/MULTI SITE	X			X	X				X	X	X	X
	SUPPORT SOFTWARE MATURITY									X		X	
CALIBRATION FACTOR				X		X				X			
EFFORT EQUATION	$MM_{NOM} = C(DSI)^X, X =$		1.0		1.047		0.91	1.0		1.08 - 1.2		1.0	1.2
SCHEDULE EQUATION	$S_D = C(MM)^X, X =$						0.36			0.32 - 0.38		0.366	0.333

Figure 3.2 Factors Used in Various Cost Models

#### IV. EXISTING COST ESTIMATION MODELS

##### A. BACKGROUND

The following is a brief introduction of several models that have been developed for software cost estimation. In presenting these models it is important to note that the majority of the models have been developed based on a particular set of data and environmental factors. Therefore, most of the models are not transportable unless they provide a method for recalibration. Furthermore, a majority of the models require complex mathematical calculations that are very cumbersome for the user to apply to their own situations.

There are a few models, however, that have been automated and are commercially available. These include the SLIM, PRICE S, ESTIMACS and SOFTCOST models. Bailey et al. [Ref. 22] provides a detailed evaluation of many of the automated software cost-estimation models currently available.

Further detailed explanation of each model may be found in the original source reference listed for each model. An overview of a majority of the models is also provided by Boehm [Ref. 19:pp. 510-520], Londeix [Ref. 3:pp. 36-41] and Thibodeau [Ref. 4]. The COCOMO and Function Point models are described in detail in Chapters V and VI respectively and are not covered in this section.

## B. COST ESTIMATION MODELS

### 1. SDC Model

The SDC model was developed from a study completed by the System Development Corporation (SDC) for the U.S. Air Force in the mid-1960's [Ref. 23]. This study included an extensive analysis of 104 attributes of 169 software projects. The SDC model proved to be highly inaccurate as a cost estimator. It raised serious doubts about the ability of a linear cost estimation model to estimate cost. However, it did provide valuable information and spurred new research into the area of cost estimation.

### 2. TRW Wolverton Model

The TRW Wolverton model is a matrix-based model which was developed for use at TRW [Ref. 24]. In the model, estimates of routine size are converted to costs using cost per instruction values that are functions of the routine type and complexity. A matrix of ratios is used to allocate the total cost to seven phases with each phase divided into up to 25 activities.

The essence of the model can be seen in Figure 4.1, which is an example using the 6 categories of the model. The chart demonstrates how the cost per object instruction is related to the relative degree of difficulty. Relative degree of difficulty is determined by whether a routine is considered old or new and whether it is classified as easy, medium or hard.

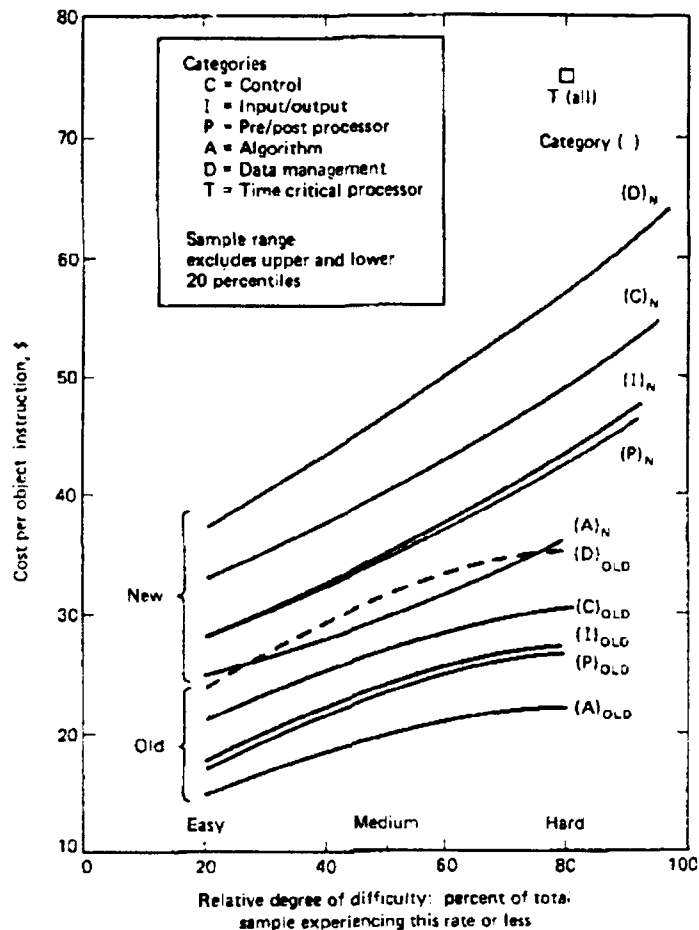


Figure 4.1 TRW Wolverton Model: Cost Per Object Instruction Vs. Relative Degree of Difficulty [Ref. 19:p. 513]

By multiplying the cost per instruction for each routine by its number of object instructions and summing the products for all of the routines, an estimated value for total development cost may be obtained. This cost is then allocated to each of the seven phases of development and their attributes as defined by the model.

It is important to note that the ratios developed in the TRW Wolverton model are only applicable for the TRW database. Therefore, new ratios and a new matrix would have to be developed if a different environment was to be used.

### 3. Putnam (SLIM) Model

The Software Life Cycle Model (SLIM) is a commercially available costing model developed by Quantitative Software Management, Inc. [Refs. 25; 26]. It is primarily based on the estimating model developed by Larry Putnam in the late 1970s. The SLIM model depends on a SLOC estimate for the project's general size. It also uses formulas which relate software size to cost and schedule requirements. Therefore it is not exactly a pure form of the phenomenological model as described by Thibodeau [Ref. 4]. However, a major portion of the model is dependent upon relationships which follow the scientific Rayleigh distribution curve. In particular, the model relates the software life cycle to the Rayleigh curve.

A majority of the articles regarding the Putnam model and how it incorporates the Rayleigh distribution curve may be found in [Ref. 27]. Weiner-Ehrlich et al. [Ref. 28] provides an additional source of information regarding the use of the Rayleigh curve for software modeling.

The Slim model uses the following equation in developing its estimation model:

$$S_s = C_k K^{1/3} \times t_d^{4/3}$$

where:

$S_s$  = source statements (code size in SLOC)

$C_k$  = technology constant (dimensionless usually 10040)

$K$  = life cycle effort in man-years

$t_d$  = development time in years.

The SLIM model combines the estimation equation with Monte Carlo simulation, standard deviation analysis, and Rayleigh/Norden distribution curve analysis to provide a unique estimate of effort and development time. One of the basic assumptions of this model is that manpower utilization during program development follows the Rayleigh curve. Therefore, manpower and cash flow rate may be obtained at any point in the life cycle.

SLIM provides two methods for properly calibrating the model: the user can calibrate the model by either inputting data from completed projects, or by answering a series of 22 questions from which Slim will provide recommended calibration entries.

#### 4. Doty Model

In 1977, Doty Associates Inc., produced a software cost estimation study of the software developed for the RADC (Rome Air Development Center) [Ref. 29]. The objective of the study was to reduce the variance between the estimated and

actual cost of software development. The study resulted in the development of the Doty model.

The model is actually a set of individual models. Each one to be used for a given type of software environment. Equations have been developed empirically using regression analysis for four application areas which include: command and control, scientific, business and utility. These equations use size inputs to estimate the number of man-months of effort required for the program as defined by its category type. A general set of equations is also available for programs which do not fit into any of the four predefined categories. The model also uses a series of 14 effort multipliers in order to better refine the cost estimate to the development environment. The model is considered a static, multivariable model.

##### 5. RCA PRICE S Model

The PRICE S model is an automated proprietary cost-estimation model developed and maintained by PRICE Systems Division of RCA, New Jersey [Ref. 30]. It is currently available through an On-line system which can be reached by modem over a standard telephone line. The model was primarily designed for aerospace applications. It is considered a macro estimation model as it uses a top-down approach throughout its estimation development.

PRICE S uses inputs of size, type and difficulty and a series of hypothesized relationships in order to make



estimates of project cost and schedule. Most inputs are heuristic in nature as they are based on certain subjective opinions of the user, such as:

- the cost required to produce programs,
- the effect on cost of changing development time,
- the comparative costs of the development cycle elements.

Since every project is different and will take in different inputs, the model does not have a standardized set of equations with predetermined coefficient values for calculating effort.

#### 6. Walston and Felix IBM-FSD Model

This model was developed by Walston and Felix at IBM Federal Systems Division in an attempt to measure the rate of production of lines of code by project as influenced by a number of product conditions and requirements [Ref. 31]. The model is derived from a database of 60 different projects. One of the goals of Walston and Felix was to develop an effort estimation model based on size alone. Based on their collected data they found 29 factors which were significantly correlated with productivity. They incorporated these factors into a single formula which enabled them to calculate a productivity index. Using this productivity index and linear regression (for calibrating their model to the environment), they developed an equation for estimating productivity of new projects.

## 7. Boeing Model

The Boeing model was developed by Boeing Computer Services in 1977 [Ref. 32]. The model is considered to work best for aerospace types of systems for which it estimates total project effort. In estimating effort the model uses a set of productivity rates applied to the following types of software used in the model:

- Mathematical Operations,
- Report Generation,
- Logic Operations,
- Signal Processing or Data reduction,
- Real Time.

The Boeing model also uses estimates on the total number of delivered instructions for developing its effort estimation of nominal man-months. Like many other models, it uses predetermined percentages to divide the total estimated effort into individual effort estimates for the various life cycle phases. Finally, the model applies effort multipliers to the nominal effort estimates for each phase to produce an adjusted effort estimate for each phase.

## 8. GRC Model

The General Research Corporation (GRC) model was developed in 1974 [Ref. 33]. The model is a static, single-variable model which estimates cost in a non-linear fashion. The model uses a large number of different estimating techniques including regression analysis. The model has a

number of good features which include a thorough definition of the quantities being estimated and a set of relationships for estimating such quantities as training and installation costs. However, it does have a few problem areas the least of which includes using the "number of output formats" as the basic size parameter [Ref. 19:p. 519].

#### 9. Other Models

In addition to the previously listed models there have been many other software cost estimation models recently developed, some of these include the Bailey-Basili Meta-Model [Ref. 34], Grumman SOFCOST model [Ref. 35], Tausworthe Deep Space Network (DSN) model and subsequent SOFTCOST model [Ref. 36], Jensen model [Refs. 37; 38], Estimacs model [Refs. 39; 40], SPQR/20 model [Ref. 41], Before You Leap (BYL) model [Ref. 42], and the BIS/Estimator model [Ref. 43].

Most of these models have been automated and use either SLOC or function points as their primary input size metric. Since the automated models are continually under revision, the software vender should be contacted for the latest information regarding the model.

#### C. SUMMARY

Although there are many models currently available for estimating software cost, a model has yet to be developed which can estimate software cost with a high degree of accuracy. Furthermore, none of the discussed estimation

models appear to be conducive to DSS software. Nevertheless, the COCOMO [Ref. 19] and Function Point [Refs. 17; 18] models have proved to be key models in leading the research for the development of better cost estimation models. Therefore, they lend themselves as the best candidates to be tailored to the DSS environment. These two models will be presented in the following two chapters.

## V. COCOMO MODEL

### A. OVERVIEW

The COCOMO model which stands for CONstructive COST Model was developed by Barry Boehm and is covered in great detail in [Ref. 19]. Based on his analysis of 63 software development projects, Boehm developed a model that relates SLOC inputs to effort. The COCOMO model consists of three separate forms of the model: Basic, Intermediate, and Detailed. Each model is further broken down into three modes of software development: organic, semidetached and embedded. These modes are used to identify the development environment and general characteristics of a software project such as size and complexity.

The ICEM model, presented in Chapter VII, will automate the functions of the Basic and Intermediate models. Since the Detailed model will not be implemented in ICEM it is not discussed in this chapter.

### B. BASIC MODEL

The Basic COCOMO model is used to make quick, early, rough order of magnitude estimates of small-to-medium-sized software projects. The Basic model uses an estimated number of thousands of delivered source instructions (KDSI), and the development mode to estimate the development time and cost of a software development project.

Source instructions are defined as executable lines of code which include variable declarations, format statements and job control language but not comment statements [Ref. 19:p. 59]. All COCOMO models rely on fairly accurate estimates of KDSI in order to make accurate estimates.

As previously mentioned, the development mode of a project is determined by its characteristics such as size, complexity, and design environment. A summarized list of Boehm's criteria [Ref. 19:pp. 78-82] for the different modes follows:

- ORGANIC MODE

- \* Generally stable development environment,
- \* Minimal need for innovation in architectures of algorithms,
- \* Relatively low premium on early completion of the project,
- \* Relatively small size, usually not greater than 50 KDSI,
- \* Small experienced software development teams used,
- \* Loose coupling with external systems.

- SEMIDETACHED MODE

- \* Mixture of organic and embedded characteristics,
- \* Intermediate level of experience with related systems,
- \* Wide mix of experienced and inexperienced people,
- \* Some experience with aspects of system under development,
- \* Software project range usually not greater than 300 KDSI.

- EMBEDDED MODE

- \* Software development within tight constraints such as time and cost,
- \* Integral part of some larger system, heavily embedded and strongly coupled to it,
- \* Numerous interface requirements,
- \* High required reliability,
- \* Requires much innovation.

The Basic COCOMO effort estimating equations as separated by mode are as follows:

Organic Mode             $MM = 2.4(KDSI)^{1.05}$

Semidetached Mode     $MM = 3.0(KDSI)^{1.12}$

Embedded Mode         $MM = 3.2(KDSI)^{1.20}$

where:

MM    = man-months of development effort

KDSI = thousands of delivered source instructions.

These equations are used primarily to obtain an estimated number of man-months required for project development. Labor cost is not directly calculated due to price variances among organizations. However, labor cost may easily be obtained by multiplying the man-month values by an appropriate average man-month salary. It is recommended that the average man-month salaries be calculated separately for each major phase.

If desired an hourly rate can be determined by setting a man-month equal to 150 man-hours per month. [Ref. 19:p. 59]

The development period covered by COCOMO, for which the above equations apply, begins at the beginning of the product design phase and ends at the end of the integration and test phase. The COCOMO model provides a method for dividing total cost among these phases and additionally provides equations for estimating annual software maintenance cost. These issues are covered in detail in [Ref. 19] and will not be discussed further.

In addition to providing man-month estimates the Basic COCOMO model also provides Development Time or (TDEV) estimates. TDEV represents the number of months required for project completion. It is often referred to as development schedule and is calculated from the following formulas:

$$\text{Organic Mode} \quad TDEV = 2.5(MM)^{0.38}$$

$$\text{Semidetached Mode} \quad TDEV = 2.5(MM)^{0.35}$$

$$\text{Embedded Mode} \quad TDEV = 2.5(MM)^{0.32}$$

where:

TDEV = development time in months

MM = man-months as previously calculated.



### C. INTERMEDIATE COCOMO

One of the drawbacks in using the Basic model is that it is limited in accuracy because it does not take into account many factors which can effect software cost. Some of these factors include differences in hardware constraints, personnel quality and experience, and use of modern tools and techniques. The Intermediate COCOMO model was developed to incorporate these and other project attributes which are known to have a significant influence on software cost. The intent of adding these factors is to improve the accuracy of the model. The Intermediate model uses 15 cost drivers to make these adjustments. These cost drivers are grouped into four categories:

- Product attributes
  - \* RELY--required software reliability,
  - \* DATA--data base size,
  - \* CPLX--product complexity.
- Computer attributes
  - \* TIME--execution time constraint,
  - \* STOR--main storage constraint,
  - \* VIRT--virtual machine volatility,
  - \* TURN--computer turnaround time.
- Personnel attributes
  - \* ACAP--analyst capability,
  - \* AEXP--applications experience,
  - \* PCAP--programmer capability,

- \* VEXP--virtual machine experience,
- \* LEXP--programming language experience.
- Project attributes
  - \* MODP--modern programming practices,
  - \* TOOL--use of software tools,
  - \* SCED--required development schedule.

Each cost driver is ranked on a scale indicating its importance to a particular product. Figure 5.1 displays the ranking scale and their corresponding values with respect to the various cost drivers. Boehm [Ref. 19:pp. 119-122] explains how to properly rank each cost driver.

Once all of the values for the cost drivers are obtained they are multiplied together to obtain a single product called the Effort Adjustment Factor (EAF). This factor is then applied to the effort equation to obtain an adjusted man-month calculation.

The development modes for the Intermediate model are the same as those for the Basic model. However, the effort equations vary slightly from the Basic model and are as follows:

Organic Mode	$MMn = 3.2(KDSI)^{1.05}$
Semidetached Mode	$MMn = 3.0(KDSI)^{1.12}$
Embedded Mode	$MMn = 2.8(KDSI)^{1.20}$

where:

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
<b>Product Attributes</b>						
RELY Required software reliability	.75	.88	1.00	1.15	1.40	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product complexity	.70	.85	1.00	1.15	1.30	1.65
<b>Computer Attributes</b>						
TIME Execution time constraint			1.00	1.11	1.30	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility*		.87	1.00	1.15	1.30	
TURN Computer turnaround time		.87	1.00	1.07	1.15	
<b>Personnel Attributes</b>						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Applications experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience*	1.21	1.10	1.00	.90		
LEXP Programming language experience	1.14	1.07	1.00	.95		
<b>Project Attributes</b>						
MODP Use of modern programming practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

\*For a given software product, the underlying virtual machine is the complex of hardware and software (OS, DBMS, etc.) it calls on to accomplish its tasks

Figure 5.1 Software Development Effort Multipliers  
[Ref. 19:p. 118]

MMn = Nominal man-months of development effort.

The effect of the cost drivers is factored into the effort equation by multiplying the nominal man-months by the EAF:

$$\text{MMadj} = \text{MMn} * \text{EAF}$$

where:

MMadj = man months adjusted.

The schedule formulas by mode are the same as for the Basic model.

#### D. CALIBRATING THE COCOMO MODEL

The term calibration is used to mean that new coefficients or multipliers for an existing model are established or modified such that the same model structure applies to a database or an individual system corresponding to an environment other than the one upon which the model was developed.

In general, the COCOMO model was developed for most software cost estimations situations. However, by calibrating the COCOMO model to the user's specific environment the accuracy of the model can be greatly increased. Boehm [Ref. 19:pp. 524-528] provides two ways to calibrate the COCOMO model. The easiest way is to first determine the most appropriate development mode to be used. Then a least-squares approximation technique is used to recalculate the constant term (c) in the development mode's effort equation:

$$MM = c(KDSI)^b(EAF)$$

The least-squares technique produces the following equation which is used to calculate the new constant (c):

$$C = \frac{\sum_{i=0}^n MM_i Q_i}{\sum_{i=0}^n Q_i^2}$$

where:

$MM_i$  = actual man-months of effort

$Q_i = (KDSI_i)^b (EAF_i)$

$b$  = scale factor for mode

$n$  = number of projects in database.

The second method for recalibrating the COCOMO model uses a similar least-squares method for calibrating both the coefficient term (c) and the scale factor (b). These values are recalibrated to the user's environment by using the following equation:

$$\log c = \frac{a_2 d_0 - a_1 d_1}{a_0 a_2 - a_1^2}$$

$$b = \frac{a_0 d_1 - a_1 d_0}{a_0 a_2 - a_1^2}$$

where:

the quantities  $a_0$ ,  $a_1$ ,  $a_2$ ,  $d_0$  and  $d_1$  are calculated as:

$$\begin{aligned}
a_0 &= n \\
a_1 &= \sum_{i=0}^n \log(KDSI)_i \\
a_2 &= \sum_{i=0}^n [\log(KDSI)_i]^2 \\
d_0 &= \sum_{i=0}^n (MM/EAF)_i \\
d1 &= \sum_{i=0}^n \log(MM/EAF)_i \log(KDSI)_i
\end{aligned}$$

As is apparent, neither of these methods provides a way to recalibrate the cost driver rating values. Unfortunately, the only method to recalibrate the cost drivers is through trial and error and this is not recommended due to the multiplicative nature of the EAF factor.

The ICEM model presented in Chapter VII uses this least-squares method in order to recalibrate the model to the unique environment of Decision Support Systems.

## VI. FUNCTION POINT ANALYSIS MODEL

### A. BACKGROUND

In 1979, Alan Albrecht of IBM developed the method of Function Point Analysis [Ref. 17], to help measure the size of a computerized business information system. He found that he could not successfully use the SLOC method to determine size measurements which were needed as an input component for effort and productivity estimates. As an alternative to using SLOC, he developed the Function Point Analysis method. He further revised and refined his method in 1983 [Ref. 18].

As previously mentioned, there are very few cost estimation models that can be applied relatively early in the systems development life cycle. This is because they rely on metrics that can only be applied in the post-code phase of development such as SLOC. The function point metric is an exception to this rule.

By being able to make estimates early in the development process one can continue to refine the cost model throughout life cycle development. Furthermore, early estimates can improve scheduling and reduce cost. Another advantage of the function point metric is that, unlike SLOC, it is unaffected by the choice of programming languages used.

## B. CALCULATING FUNCTION POINTS

In addition to Albrecht's articles, there have been several noteworthy publications written which provide a step by step method for calculating function points. Brian Dreger [Ref. 44] provides a highly-detailed guideline for calculating function points while Roger Pressmen [Ref. 20] provides an easy to use table format. Figure 6.7 provides a similar format to be used as a worksheet for making function point calculations.

The method for calculating function points involves a four step process:

- Count the unique number of occurrences of the five user function types,
- Classify each function type according to its level of complexity,
- Adjust for processing complexity,
- Make the function points calculation.

## C. (STEPS 1 & 2) COUNT AND CLASSIFY FUNCTION TYPES

Five types of functions are counted as function points:

- Inputs.
- Outputs,
- Inquiries,
- Files,
- Interfaces.

Each of these functions are classified according to three levels of complexity:



- Simple,
- Average,
- Complex.

These complexity factors are further associated with a particular weighting factor which is used in (Step 3) to adjust the values of the five individual function counts.

#### 1. Measuring Inputs

In measuring inputs, each unique user data or control input that is performed by the user within the application in order to add, delete or update something should be counted. An input is considered unique if it has a different format, such as a different input screen, or it has the same format as another input but uses different processing logic (same entities are modified in a different way). Inputs should be distinguished from inquiries, which are counted separately. After the number of inputs are counted they are classified according to their complexity.

##### a. Classifying Inputs

Classifying inputs for complexity depends on two things: the number of files referenced or accessed (see measuring files below) and the number of data items (fields or specific variables) referenced. It is important to note that only the data items actually updated by the input transaction are counted. Data items which reside in the same file but are not referenced are not counted. The complexity level and associated weighting factor is selected by cross-referencing

the numbers of files referenced to the number of data items referenced. Figure 6.1 is used to determine the complexity level and the corresponding complexity weighting factor.

	1-4 data items referenced	5-15	16 or more data items
0 or 1 file(s) referenced	Simple (3)	Simple (3)	Average (4)
2 files referenced	Simple (3)	Average (4)	Complex (6)
3 or more files ref.	Average (4)	Complex (6)	Complex (6)

Figure 6.1 Classifying Inputs

From this chart it may be seen that ten data items accessed from two files would be classified as "average" in complexity and given a weighting factor of four.

## 2. Measuring Outputs

In measuring outputs, each unique user data or control output procedurally generated that leaves the application boundary should be counted. This includes reports and messages to the user, as well as outputs to other applications. Uniqueness has the same implications for outputs as it does for inputs.

### a. Classifying Outputs

Outputs are classified in a similar format as inputs. However, the actual numerical values for the various

entries have been changed. Only the files and individual data items accessed during output are counted. Figure 6.2 is used to obtain the complexity level and corresponding weighting factor for outputs.

	1-5 data items referenced	6-19	20 or more data items
0 or 1 file(s) referenced	Simple (4)	Simple (4)	Average (5)
2-3 files	Simple (4)	Average (5)	Complex (7)
4 or more files ref.	Average (5)	Complex (7)	Complex (7)

Figure 6.2 Classifying Outputs

### 3. Measuring Inquiries

In measuring inquiries, each unique input/output combination in which the on-line user-defined input causes and generates an immediate on-line output by the application should be counted. Inquiries may also be provided to other applications. Many inquiries are simply requests for specific data from a data base. An inquiry is considered unique if it has a format different from others in either its input or output portions or it has the same input and output format but requires different processing logic in either.

a. Classifying Inquiries

Classifying inquiries consists of two parts: classifying the inputs and classifying the outputs. The same charts are used for inputs and outputs as before. The only difference is that while a stand-alone input actually updates the data store, an inquiry only directs the search and never updates. Once the two complexity factors are obtained for outputs and inputs the two are compared with the larger value being selected as the weighting factor for the inquiry function.

Figure 6.3 is used to obtain the respective classification and weighting factor for the input part of the inquiry function, while Figure 6.4 is used to obtain the output factor. The larger value of the two is used as the factor for the inquiry function.

Input part:	1-4 data items referenced	5-15	16 or more data items
0 or 1 file(s) referenced	Simple (3)	Simple (3)	Average (4)
2 files referenced	Simple (3)	Average (4)	Complex (6)
3 or more files ref.	Average (4)	Complex (6)	Complex (6)

Figure 6.3 Classifying Inquiries (Input)

Output Part:	1-5 data items referenced	6-19	20 or more data items
0 or 1 file(s) referenced	Simple (4)	Simple (4)	Average (5)
2-3 files	Simple (4)	Average (5)	Complex (7)
4 or more files ref.	Average (5)	Complex (7)	Complex (7)

Figure 6.4 Classifying Inquiries (Output)

#### 4. Measuring Files

Measuring the number of files is not as simple as just counting the number of physical files in an application. Rather, only files that contain data stored in logical groupings within the application are counted as files. These files perform data storage functions for the application. Furthermore, files or data stores are to be considered in the logical not physical sense. A physical file can actually contain many logical files. Every unique data access, path or view of a database is considered a collection of information and is counted as a separate logical internal file. However, temporary data stores are excluded from this count as only permanent files are counted. Additionally, transactions that trigger internal logical files to be updated or changed are not considered files themselves.

It is crucial that the number of logical internal files be properly counted. Typically, every logical file will

have at least one input, output, and inquiry. This corresponds to at least 18 points when complexity adjustments are added (seven file, three input, four output, and four inquiry).

#### a. Classifying Files

Like the other classifications, classifying files is a two-step process. First the number of data items actually required by the application are counted and then either the number of record formats within the file or the number of logical relationships in which the file participates are counted. It is important to recognize that only logical relationships are used; therefore, the number of different record types within a file are not simply counted but also their logical relationships as well. Figure 6.5 is used to determine the complexity level and weighting factor for files.

	1-19 data items ref.	20-50	51 or more data items
1 logical record format/relation- ship	Simple (7)	Simple (7)	Average (10)
2-5 logical record format/ relationships	Simple (7)	Average (10)	Complex (15)
6 or more logical record format/relation- ships	Ave. (10)	Complex (15)	Complex (15)

Figure 6.5 Classifying Files

## 5. Measuring Interfaces

Interfaces involve using data stored by another application but used by the current application. In measuring interfaces count every major logical file (as previously defined) within the application boundary that is sent to, shared with, or received from another application. Files shared between applications are counted as both files and interfaces within each application if they are used in both. This includes data stores that are imported, exported or shared between the two applications. Interface does not involve transaction. An application must be able to access the data directly without the aid of another application for it to be counted as an interface.

### a. Classifying Interfaces

The classification scheme used for interfaces is similar to the one used for files. As Figure 6.6 demonstrates the number of data items referenced (and actually used) and the number of logical relationships in which the interface file participates to meet application requirements are used to obtain the complexity level and weighting factor.

	1-19 data items ref.	20-50	51 or more data items
1 logical record format/relationship	Simple (7)	Simple (7)	Average (10)
2-5 logical record format/ relationships	Simple (7)	Average (10)	Complex (15)
6 or more logical record format/relationships	Ave. (10)	Complex (15)	Complex (15)

Figure 6.6 Classifying Interfaces

D. (STEP 3) ADJUST FOR PROCESSING COMPLEXITY

Adjusting for processing complexity is a simple task; simply multiply the measured value for each function (count) by its corresponding weighting factor. Figure 6.7 provides a worksheet for developing function points and shows how the weighting factor is incorporated into the process of calculating the function point value.



Function	Count	Weighting factor			FP
		Simple	Avg.	Complex	
Inputs		3	4	6	
Outputs		4	5	7	
Inquiries		3	4	6	
Files		7	10	15	
Interfaces		5	7	10	
Count X weighting factor = FP			Sum of all FP =		

Figure 6.7 Function Point Worksheet

E. (STEP 4) MAKE THE FUNCTION POINTS CALCULATION

To make the total adjusted function points calculation (FP) the following equation is used:

$$FP = (\text{Sum of FP counts}) \times [0.65 + (0.01 \times \text{SUM}(F_i))]$$

where (Sum of FP counts) is the total sum of FP counts obtained from above and the SUM( $F_i$ ) is the sum of 14 complexity adjustment values (where  $i = 1-14$ ) obtained by answering the questions in Figure 6.8 according to the ranking scale provided.

---

Rate each factor on a scale of 0 to 5:

0	1	2	3	4	5
No influence	Incidental	Moderate	Average	Significant	Essential

Fi:

1. Does the system require reliable backup and recovery?
  2. Are data communications required?
  3. Are there distributed processing functions?
  4. Is performance critical?
  5. Will the system run in an existing, heavily utilized operational environment?
  6. Does the system require on-line data entry?
  7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
  8. Are the master files updated on-line?
  9. Are the inputs, outputs, files, or inquiries complex?
  10. Is the internal processing complex?
  11. Is the code designed to be reusable?
  12. Are conversion and installation included in the design?
  13. Is the system designed for multiple installations in different organizations?
  14. Is the application designed to facilitate change and ease of use by the user?
- 

Figure 6.8 Complexity Adjustment Factors

#### F. SUMMARY

The four step method we have followed has only one goal in mind, to calculate a single function point number. It is important to note that this number is only a representation of the size of the project. Like SLOC it is only used as a sizing metric and therefore does not yield an effort estimation directly. However, as demonstrated by Albrecht and Gaffney in 1983 [Ref. 18], one can use simple linear regression on a data set of projects to estimate man-months as a function of the function points. This is the method used in developing the ICEM model.

## VII. INTELLIGENT COST ESTIMATION MODEL (ICEM)

### A. INTRODUCTION

The ICEM model is an integrated automated package which utilizes an expert system coupled with a spreadsheet to perform cost estimation. The model was developed as an initial prototype system. It is not intended to be an all-inclusive model ready to be implemented within an organization. Rather, it is used to analyze existing methodologies and promote the development of integrated cost estimation models which incorporate expert system technology.

One of the key factors in accurately estimating cost is user experience. All too often this experience is lost when personnel transfer. The ICEM model is considered intelligent because it uses an expert system to collect and process heuristic data. An expert system is used because it enables an organization to capture the talent and experience of its key personnel [Ref. 45:p. 332]. By using an expert system as the foundation for the model we hope to promote the capturing of this experience so it can be used in designing and continually upgrading an effective cost estimation model.

Additionally, the expert system enables the quick development of a very user-friendly system. The expert system uses a simple question and answer format to collect data for its parametric models. This eliminates the need for the user

to perform exhaustive searches through data tables to obtain appropriate data values which must then be applied to equations for manual calculations.

The main problem with almost all of the cost estimation models discussed in Chapter IV is that they are poor estimators. This point was emphasized by Boehm [Ref. 19:p. 32] who said:

Today, a software cost estimation model is doing well if it can estimate software development cost within 20% of the actual costs, 70% of the time, and on its own home turf (that is, within the class of projects to which it is calibrated).

Most of this low accuracy rate is due to undersizing. But, even as weak as these cost estimation models may appear, they still offer marked improvements over previously used manual methods and if properly used can be very beneficial.

In an effort to improve the accuracy of the estimating process the ICEM models incorporates the concepts developed by the COCOMO and Function Point models into an integrated model. These two models were selected based on their popularity and success [Ref. 5]. Also these two models proved ideal because they utilize two very dissimilar estimating methods and incorporate two different size metrics (SLOC and Function Points) for their primary input. For the purpose of consistency checking it is important to have as much independence in the estimating methods as possible.

The ICEM model is actually three models in one. The COCOMO Intermediate model has been automated for all modes of

operation including organic, semidetached and embedded modes. The second model specifically tailors the COCOMO organic mode for the DSS environment. The organic mode was considered because of the characteristics of the empirical database which is used to calibrate the model. The third model includes a parametric model which incorporates the Function Point size metric.

In order for any cost estimation model to be useful in an environment other than the one in which it was developed, it must be calibrated to the new environment. One of the requirements for tailoring a cost estimation model to a new environment is to have a proper database. To ensure statistical quality, a database of at least ten projects unique to the new environment is recommended as a starting point. The ICEM model is calibrated to the Decision Support System (DSS) environment by using a database set of 13 DSS projects.

#### B. OVERALL ARCHITECTURE OF ICEM

The architecture of the ICEM model is displayed in Figure 7.1. **VP-Expert**, developed by Paperback Software International, provides the primary interface with the user. Although several expert systems are currently on the marketplace today, **VP-Expert** was selected based on its low cost, ease-of-use, powerful expert system capabilities, and ability to be easily

coupled with spreadsheet (VP-Planner) and database (VP-INFO) systems.

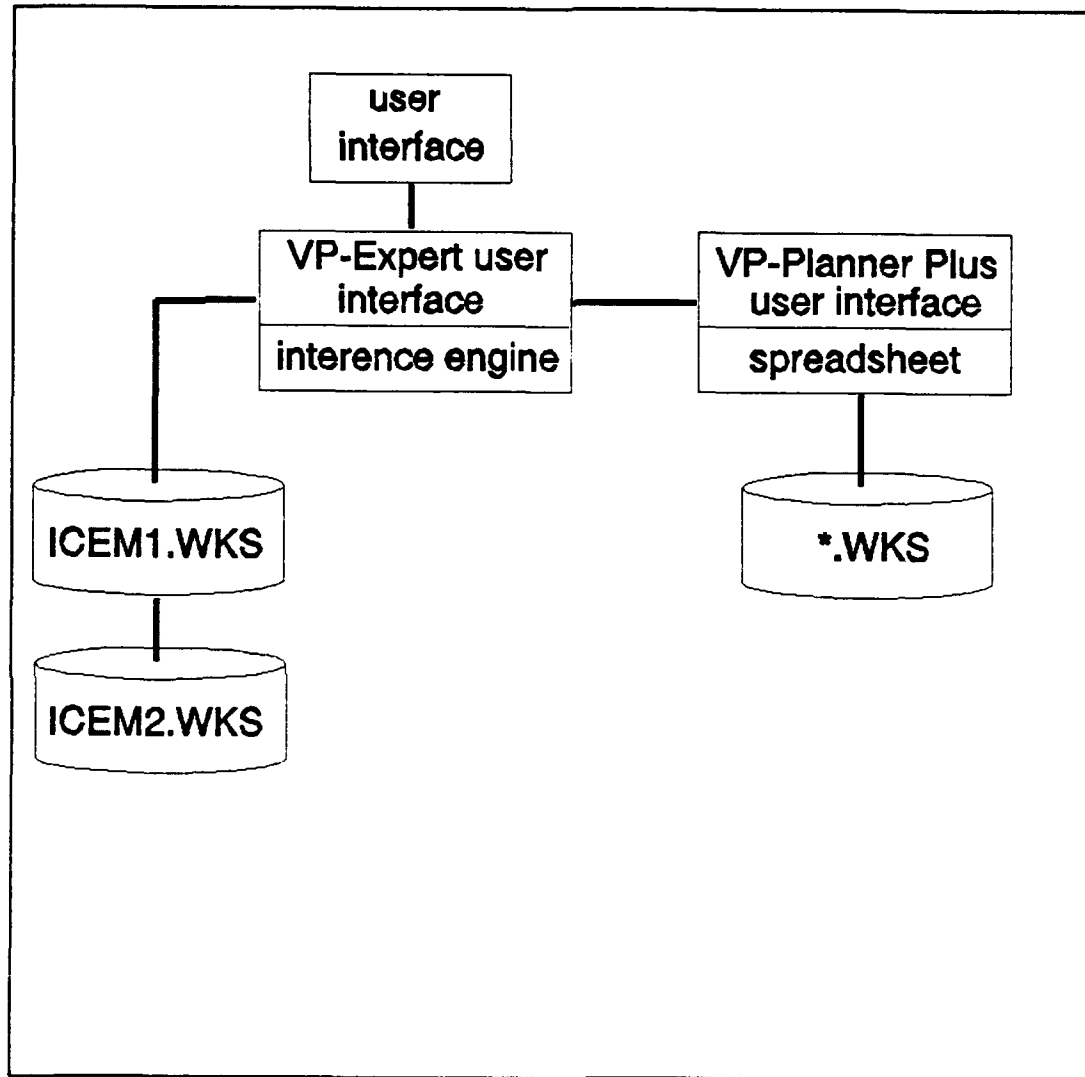


Figure 7.1 ICEM Architecture

1. VP-Expert

VP-Expert consists of three parts: a user interface, a knowledge base, and an inference engine. The user

interface allows the user to communicate with the expert system through the keyboard and screen display. In VP-Expert, the knowledge base is stored in files that have the extension ".KBS." The knowledge base itself is composed of three sections: actions block, rules block, and questions block.

The actions block of the knowledge base provides directions to the expert system for finding a particular solution to a problem. It controls the flow of the program. On the other hand, the rules block contains the rules that tell VP-Expert how to solve a specific problem. It consists of three key words: RULE, IF, and THEN. The questions block provides questions to ask the user if VP-Expert needs more information.

Finally, the inference engine contained in VP-Expert provides decision-making intelligence. The inference engine uses the rules in the rule base in order to make decisions on how to solve a problem. Typically, an expert system's inference engine can use two methods for processing the rule base, either forward chaining or backward chaining. The ICEM model was developed to employ a backward chaining strategy for problem solving.

The ICEM model incorporates two knowledge bases, ICEM1.KBS and ICEM2.KBS, which have been chained together. One of the main problems with using a rule-based expert system is that depending on the size of the rule base the system could require a considerable amount of memory usage. In order



to get around this memory barrier VP-Expert allows two rule bases to be chained together.

In VP-Expert, when a knowledge base is chained to a second knowledge base, variable values are first stored in a temporary file using the "SAVEFACTS" command. The second knowledge base (with its new rule base) then replaces the first rule base in memory. Finally, it recovers the stored variable values using the "LOADFACTS" command. This method saves considerable memory by swapping knowledge bases and their accompanying rule bases.

One minor inconvenience of using the "SAVEFACTS" command is that it will save the value of every variable used by any portion of the expert system resulting in irrelevant data passing from one knowledge base to another. In order to eliminate unnecessary data passing all variable values can be cleared from memory using the "RESET ALL" command and then pertinent variables can be assigned specific values before using the "SAVEFACTS" command. By using the "RESET ALL" command and chaining its two knowledge base files (ICEM1.KBS and ICEM2.KBS) together, the ICEM model is able to save considerable memory thus enabling it to operate within the boundaries of conventional memory. Further information about VP-Expert and its command language may be found in [Ref. 46].

## 2. VP-Planner Plus

As shown in Figure 7.1, VP-Expert is coupled to VP-Planner Plus. VP-Planner Plus is a spreadsheet program also

developed by Paperback Software International. It is used to provide automated statistical calculations and display effort and schedule information in a format easily understood by the user. It is fully automated and allows instantaneous sensitivity analysis to be performed by the user. All variable values are passed from VP-Expert to VP-Planner using the "PWKS" command. Although this command works relatively well, it has a tendency to be time-consuming. It takes approximately 20-30 seconds to save values to the spreadsheet using an IBM compatible 386 while a 286 machine may take one to two minutes for processing.

Although there are many advantages to linking VP-Expert to VP-Planner there is also one additional disadvantage. When the two programs are linked together they are essentially running simultaneously. Therefore, almost twice the amount of memory is required than if only one of the programs was running at a time. This posed a significant challenge in trying to keep the memory usage under 600k. Unfortunately, there is no way around this problem. VP-Expert does not offer a method to call a specific spreadsheet using VP-Planner unless VP-Expert is running in the background.

All user data are saved in spreadsheet files that end with the ".WKS" extension. Further information about VP-Planner Plus and its related commands may be found in [Ref. 47].

## VIII. USING ICEM FOR EFFORT ESTIMATION OF DSS SOFTWARE

### A. INTRODUCTION

The purpose of this chapter is to use ICEM as a DSS in order to estimate the amount of effort required to develop PC-based DSS. As discussed earlier, ICEM combines the COCOMO model and the Function Point models. But before these models can be incorporated into ICEM, it is first necessary to recalibrate their estimated coefficients to take into consideration environmental factors related to DSS.

As can be seen later in this chapter, recalibration is particularly necessary, especially when you consider that the original coefficients developed by Boehm for the COCOMO model were derived from large-scale projects using second generation programming language (2GL) that were developed two decades ago. On the other hand, the majority of DSS are developed by small programming groups using either third or fourth generation programming languages (4GL).

Although ICEM was designed to incorporate both the COCOMO and function points methodologies, only the COCOMO model has been calibrated due to the unavailability of empirical data.

### B. THE DSS DATABASE

The DSS database, which the ICEM model uses for calibration, comprises 13 DSS projects which were built at the Naval

Postgraduate School (for more information regarding these projects, contact the thesis's advisor). All of these DSS were developed under the conditions described by the COCOMO organic mode (see definition, Chapter V, Section D). Table 8.1 reproduces the data points for the various DSS projects. A brief description of each of the 13 DSS project follows.

TABLE 8.1  
DSS DATA POINTS

Project	Language	KDSI	EAF	MMact	#dev
CO-OP	Turbo Pascal	16.0	0.94	15.0	1
INTEG	Basic	3.0	1.15	7.0	2
CEA	Turbo Pascal	9.0	0.90	7.0	1
TAO	Exsys	4.0	0.85	6.0	1
CEASAR	Exsys	5.0	0.98	5.0	1
NURSE	Turbo Pascal	5.0	1.26	9.5	1
ASDB	dBase III	6.0	1.00	6.5	2
COCOMO	Knowledgeman	3.0	1.30	7.5	2
NAVAIR	dBase III	2.5	0.90	6.5	3
CAI	dBase III	2.5	1.50	8.0	2
STOCKPT	VP-Expert	6.0	1.00	4.0	1
DIST.ES	VP-Expert	6.5	1.10	3.0	1
TOUCHSTONE	Turbo Pascal	9.5	1.00	8.0	2

#### 1. Co-Op

Co-Op is a group DSS for multiple-criteria decision making. Co-Op contains a set of techniques of aggregation of preferences and consensus seeking algorithms that can be used in conjunction with individual multiple criteria decision models.

2. INTEG

INTEG is a Software package to assist in the instruction of an introductory graduate level course in probability and statistics. INTEG is designed to increase student productivity during time spent on learning various problem- solving techniques.

3. CEA

CEA is a DSS for cost-effectiveness analysis for control and security of computer systems. CEA is geared to help the EDP manager: (i) identify alternative sets of control activities, (ii) evaluate and choose the most preferred set, and (iii) monitor and upgrade the security of EDP system frequently.

4. TAO

TAO is a rule-based system to help Tactical Action Officers (TAO) to assess the threats of enemy's weapon systems and to determine appropriate counter measures during a naval engagement.

5. CEASAR

CEASAR is an expert system to computerize the manual assignment selection system for Army commissioned officers at the Military Personnel Center (MILPERCEN). The system is designed to minimize adversary relationships that often exist between officers in the field and their assignment specialists from the U.S. Army Military Personnel Center.

6. NURSE

NURSE is an expert system to automate the Nursing Diagnosis, Nursing Care Plan and Patient Classification Level. NURSE passes this information to another program to determine nursing staffing.

7. ASDB

ASDB is a DSS to support the management and accountability of the property of an academic department. ASDB is an intelligent DBMS that provides customized reports including custodian listings, quarterly reports, and property reports.

8. COCOMO

COCOMO is a DSS to perform sensitivity analysis of the COCOMO models including phase distribution calculation for development or maintenance, activity distribution by phase for development, and report generation.

9. Navair

Navair is an automated evaluation tool to estimate Aircraft System Test and Evaluation (AST&E) efforts. A relational DBMS is coupled with a statistical software package to estimate AST&E cost drivers and physical/performance characteristics.

10. CAI

CAI is an intelligent computer-aided instruction software system based on the Bayesian Probabilistic Model.

The system is able to function beyond the usual stand-alone mode through interfacing with an external DBMS.

11. Stock Point Expert System

The Stock Point Expert System is an expert system for causative research in inventory management. Four technical areas of causative research are implemented using four separate knowledge bases. The Stock Point Expert System seeks to improve productivity and assists with training in the causative research area of inventory management.

12. Distributed Expert System

The Distributed Expert System is a distributed expert system to provide the submarine Ship's Duty Officer (SDO) preventive maintenance expertise for the safe and effective execution of all maintenance aboard ship. The preventive maintenance knowledge is drawn from a variety of sources of expertise stored in different knowledge bases that are physically dispersed in a network of personal computers.

13. Touchstone

Touchstone is a criteria development program for group DSS. Based on the Delphi brainstorming technique, Touchstone is a text-based GDSS to help group members generate problems and explore solutions.

C. CALIBRATING THE COCOMO MODEL FOR DSS

Using ICEM and its 13-project DSS database, the calibrated Intermediate COCOMO equation is shown below:

$$MM = 1.69 \text{ KDSI}^{.75} * \text{EAF}.$$

Table 8.2 reproduces the estimated efforts (MMest) and the adjusted effort values (MMadj), using the calibrated Intermediate COCOMO model incorporating the above equation. The Effort Adjustment Factors (EAFs) were derived from close observations of the projects. Detailed description of the conditions under which these software were developed can be found in the related technical reports or theses (for more information regarding the computation of the cost drivers, contact the thesis' advisor).

TABLE 8.2  
EFFORT ESTIMATES USING CALIBRATED INTERMEDIATE  
ORGANIC COCOMO MODEL

Project	KDSI	EAF	MMest	MMadj	%ERR
CO-OP	16.0	0.94	13.72	12.89	-14.00
INTEG	3.0	1.15	3.88	14.46	-36.00
CEA	9.0	0.90	8.89	8.00	14.00
TAO	4.0	0.85	4.82	4.10	-31.60
CEASAR	5.0	0.98	5.70	5.59	11.80
NURSE	5.0	1.26	5.70	7.18	-24.40
ASDB	6.0	1.00	6.54	6.54	.62
COCOMO	3.0	1.30	3.88	5.04	-32.80
NAVAIR	2.5	0.90	3.38	3.04	-62.00
CAI	2.5	1.50	3.38	4.22	5.50
STOCKPT	6.0	1.00	6.54	6.54	118.00
DIST.ES	6.5	1.10	6.59	7.65	-4.30
TOUCH	9.5	1.00	9.26	9.26	42.40
% Mean Error = -1.0% Sum of the squared errors = 72.22					



Table 8.3 reproduces these efforts using Boehm's Intermediate Organic COCOMO model which has not been calibrated to the DSS environment.

TABLE 8.3

EFFORT ESTIMATES USING NON-CALIBRATED INTERMEDIATE  
ORGANIC COCOMO MODEL

Project	KDSI	EAF	MMest	MMadj	%ERR
CO-OP	16.0	0.94	58.81	55.28	268.53
INTEG	3.0	1.15	10.16	11.66	66.57
CEA	9.0	0.90	32.14	28.93	313.29
TAO	4.0	0.85	13.72	11.66	94.33
CEASAR	5.0	0.98	17.34	16.99	239.80
NURSE	5.0	1.26	17.34	21.85	130.00
ASDB	6.0	1.00	21.00	21.00	223.08
COCOMO	3.0	1.30	10.14	13.18	75.73
NAVAIR	2.5	0.90	8.38	7.54	-5.75
CAI	2.5	1.50	8.38	12.56	214.00
STOCKPT	6.0	1.00	21.00	21.00	600.00
DIST.ES	6.5	1.10	22.84	25.12	214.00
TOUCH	9.5	1.00	34.02	34.02	423.38
% Mean Error = 219.77%					
Sum of the squared errors = 4143.88					

#### D. DISCUSSIONS

Based on the data gathered in Tables 8.1, 8.2 and 8.3, a number of observations can be made:

- As expected, estimations using the calibrated model is by far much more closer to actual figures. The percentage means of errors are -1% and 220% for the calibrated and non-calibrated models respectively. The sum of the squared errors drops from 4143.88 to 72.22 when using the calibrated model.
- Actual MMs of DSS projects using software generators such as expert systems shells (i.e., VP-Expert, Exsys), data

base management systems (dBase III), or spreadsheet (i.e., Knowledgeman) are significantly lower than estimated MM. This suggest that DSS generators do help increase software productivity. The difference in using 4GL as opposed to 2GL was demonstrated by Verner and Tate, who found that using 4GL to build a DSS reduced development effort and schedule compared with Cobol in all phases of the life cycle except the requirements phase. [Ref. 48]

- The data also suggest that small-size, organic DSS, projects developed by one person appear to require less effort than those involving more than one person. The COCOMO effort adjustment factor (EAF) does not take into consideration the number of people associated in the software development. It is suspected that interpersonal communications as well as coordination and division of labor contributed to these discrepancies. However, it is not evident that there is a direct linear relationship between the number of personnel working on a project and the total MM required.
- A final comment relates to the experience in counting function points for the 13 DSS. Unlike SLOC measurements, ex-post data gathering for function points has proved to be much more difficult. It is believed that this process would have been much easier and more accurate if data had been gathered during the early phases of the software development. Additionally, the Function Point model appears to be more adapted to be used in a structured analysis setting where one can use Data Flow Diagrams (DFDs) and Entity Relationship Diagrams (ERDs) for data gathering purposes. Unfortunately, none of the projects in the DSS database incorporated structured analysis techniques in their development. This made data gathering much more difficult and inaccurate.

## IX. CONCLUSIONS AND RECOMMENDATIONS

### A. CONCLUSIONS

The purpose of this thesis was to provide a tool to facilitate the tailoring of software effort estimation tools to the small-scale, organic DSS environment. The most popular cost estimation models were reviewed in Chapters IV, V and VI. It was found that none of these techniques were conducive to a DSS software development environment.

ICEM was implemented as a workbench to calibrate the COCOMO and the function point metrics for DSS (Chapter VII). Using a database of 13 projects, a calibrated COCOMO model was derived (Chapter VIII). As expected, the study has found that the calibration was a critical condition when using parametric models. The findings also revealed some weaknesses of the COCOMO models for DSS effort estimation.

Because of the inherent inaccuracies of estimation techniques, a model that can consistently estimate the effort and cost of software development with a high degree of accuracy still does not exist. However, by using a combination of modeling tools particularly tailored to the user's environment, better estimates can be made.

Cost estimation models should not be implemented with the intent of replacing the experienced estimator. Rather they should be used as a tool to assist the estimator. They should

be used to reinforce the estimator's decisions not replace them. They can be also be used to perform sensitivity analysis and to keep track of the evolution of the cost patterns. ICEM was implemented with this concept in mind.

#### B. RECOMMENDATIONS FOR FURTHER RESEARCH

The calibrated equation for DSS proposed in Chapter VIII proved to be much more accurate than the original COCOMO equation. To further improve the accuracy of the model, the DSS database should be regularly updated and the model recalibrated. Due to environmental changes old projects in the database may have to be deleted as their information becomes obsolete or outdated while new data should be added to the database. It is recommended that ICEM be coupled with a database management system for that purpose. Additionally, if the size of the data permits, the model should be calibrated using subsets of data points according to the types of programming environment, DBMS, expert system shells, and 4GLs. By tailoring the model to the specific environment and programming language being used in the project, more accurate estimates are possible. Last but not least, the parameters used in the ICEM knowledge base to determine cost drivers were derived from large-scale projects. It is desirable that these factors also be calibrated to DSS environments to further improve the accuracy of the estimation.

## APPENDIX A

### SAMPLE SESSION

#### I. A TYPICAL ICEM CONSULTATION

##### A. OVERVIEW

This appendix is an example of a session using the Intelligent Cost Estimation Model (ICEM). The figures which follow are similar to actual screen outputs of the ICEM model as run on an IBM AT personal computer. In **VP-Expert**, the user highlights menu items to be selected. In this appendix, highlighted items are shown in **bold and underlined** type. The total run-time of a typical session is between five and ten minutes.

##### B. RUNNING ICEM

To start the ICEM model simply type **GO** and the welcome screen will be displayed, Figure A.1. If any key is depressed the next screen will be displayed, Figure A.2. The first option the user must decide is whether to retrieve a previously saved file. All previously saved files end with the extension **.WKS**, such as **DSSCOC.WKS**. Since **VP-Expert** requires the file name to be pre-specified within the knowledge base, only the following files may be retrieved: **ORGANIC.WKS**, **SEMIDET.WKS**, **EMBEDDED.WKS**, **DSSCOC.WKS**, and **DSSFP.WKS**. Since all spreadsheets are saved under these names the user is advised to copy all spreadsheets files that they would like to save to a new filename, otherwise ICEM will copy over the spreadsheet the next time it is run. If a file is saved under a different name then it may viewed by typing: **VPP <filename>**. To exit ICEM simply type **/q** when it is displayed at the bottom of the screen, depending on the user's location within the program it may be necessary to repeat the **q** or **/q** process to exit.

Since a previously saved file was not retrieved, the next question addressed to the user is which model they would like to select. There are three different models available within ICEM: the intermediate **COCOMO** model for all modes of operation, the intermediate **COCOMO** model (organic mode) which has been calibrated to the DSS environment, and a parametric model which incorporates the Function Point size metric.

For this session the **COCOMO** model calibrated for the DSS environment is selected. The next input the model requests is for an estimate of the number of thousands of lines of code the program

will have. Additionally, the COCOMO model requires 15 cost driver values which take into account factors effecting the estimation. The user must select one of the five options displayed for each of these cost drivers. Descriptions of each selection are provided to the user.

Based on the user's inputs appropriate values are retrieved from the rule base and saved to the appropriate cells of the spreadsheet. The spreadsheet is then retrieved for the user. Figure A.3, displays appropriate output values. The final estimated value of the number of man-months required to complete the project is described as MMadj. To perform sensitivity analysis the user may change the number of KDSI or any of the values of the cost drivers, all output values are instantly recalculated. Only the cost driver values specified in Figure A4 should be used. Figure A.5 displays a sample output using the Function point model.

Welcome to the ICEM model!

The ICEM model is an integrated cost estimation model which uses an expert system to automate the Intermediate Constructive Cost Estimation Model (COCOMO), developed by Barry W. Boehm and the Function Point Model, developed by Allen Albrecht. Both of these models have been tailored to the unique environment of Decision Support Systems (DSS).

Press any key to continue!

Figure A.1

The ICEM model is actually three models in one. The COCOMO Intermediate model has been automated for all modes of operation including organic, semidetached and embedded modes. The second model specifically tailors the COCOMO semidetached mode for the DSS environment. The third model tailors the Function Point model for the DSS environment.

Press any key to begin the consultation.

Figure A.2

Would you like to retrieve a previously saved file?

YES

NO

Which of the following models would you like to select?

COCOMO = Regular COCOMO

DSSCOC = COCOMO Tailored for DSS environment

DSSFP = Function Point model tailored for DSS environment

COCOMO

DSSCOC

DSSFP

This model is designed to be used as a cost estimation tool for Decision Support System projects that meet the requirements as specified by the Organic mode of the Intermediate COCOMO model. The following characteristics apply to projects which meet these requirements:

- a. Generally stable development environment.
- b. Minimal need for innovation in architectures of algorithms.
- c. Relatively small size.
- d. Relatively low premium on early completion of the project.
- e. Software project range usually not greater than 50 KDSI.
- f. Loose coupling with external systems.

This calibrated model estimates the development time and cost (in man-months) of a software project based on inputs of estimated number of thousand of delivered source instructions (KDSI), and values for 15 cost drivers. Nominal values of 1.0 may be entered for unknown cost driver information.

What is your best estimation on the number of thousands of delivered source instructions your program will have?

5

You have entered 5 kdsi.

(Press enter to continue)

Cost drivers are factors to consider in developing a model for estimating the cost of a software project. The drivers are grouped into four categories: software product attributes, computer attributes, personnel attributes, and project attributes.

Press enter to continue

The product attributes are:

RELY - required software reliability,

DATA - data base size, and

CPLX - product complexity.

Press enter to continue



The computer attributes are:

TIME - execution time constraint,  
STOR - main storage constraint,  
VIRT - virtual machine volatility, and  
TURN - computer turnaround time.

Press enter to continue

The personnel attributes are:

ACAP - analyst capability,  
AEXP - applications experience,  
PCAP - programmer capability,  
VEXP - virtual machine experience, and  
LEXP - programming language experience.

Press enter to continue

The project attributes are:

MODP - modern programming practices,  
TOOL - use of software tools  
SCED - required development schedule.

Press enter to continue

Each of these cost driver attributes determines a multiplying factor which estimates the effect of the attribute on software development. These multipliers are applied to a nominal COCOMO development effort estimate to obtain a refined estimate of software development effort.

Press enter to continue

Ratings RELY: VLOW - effect, slight inconvenience.  
LOW - easily recoverable losses.  
NOM - moderate, recoverable losses.  
HIGH - high financial loss.  
VHI AND XTRAHI - risk to human life.

Select a rating for required software reliability (RELY).

<u>VLOW</u>	LOW	NOM
HI	VHI	XTRAHI

Ratings DATA: VLOW and LOW - DB bytes/ prog. DSI < 10.  
NOM - 10 <= D/P <= 100.  
HIGH - 100 <= D/P <= 1000.  
XTRAHI - D/P >= 1000.

Select a rating for data base size (DATA).

<u>VLOW</u>	LOW	NOM
HI	VHI	XTRAHI

Ratings CPLX: VLOW - straightline code.  
LOW - straightforward nesting of structured programming  
NOM - mostly simple nesting.  
HIGH - highly nested SP operators.  
VHI - reentrant and recursive coding.  
XTRAHI-microcode level control.

Select a rating for product complexity (CPLX).

<u>VLOW</u>	LOW	<u>NOM</u>
HI	VHI	XTRAHI

Ratings TIME: VLOW, LOW - 50 % use of available execution time.  
 NOM - 50 % use of available execution time.  
 HIGH - 70 %.  
 VHI - 85 %.  
 XTRAHI - 95 %.

Select a rating for execution time constraint (TIME).

<u>VLOW</u>	LOW	NOM
HI	VHI	XTRAHI

Ratings STOR: VLOW, LOW, NOM- 50 % use of available storage.  
 HIGH - 70 %.  
 VHI - 85 %.  
 XTRAHI - 95 %.

Select a rating for main storage constraint (STOR).

<u>VLOW</u>	LOW	NOM
HI	VHI	XTRAHI

Ratings VIRT: VLOW - major change every 12 months, minor: 1 month.  
 LOW - major change every 12 months, minor: 1 month.  
 NOM - major: 6 months minor: 2 weeks  
 HIGH - major: 2 months minor: 1 week.  
 VHI, XTRAHI - major: 2 weeks minor: 2 days.

Select a rating for virtual machine volatility (VIRT).

<u>VLOW</u>	LOW	NOM
HI	VHI	XTRAHI

Ratings TURN: VLOW, LOW - interactive.  
 NOM - average turnaround < 4 hours.  
 HIGH - 4-12 hours.  
 VHI, XTRAHI - >12 hours

Select a rating for computer turnaround time (TURN).

<u>VLOW</u>	LOW	<u>NOM</u>
HI	VHI	XTRAHI

Ratings ACAP: VLOW - 15th percentile.  
 LOW - 35th.  
 NOM - 55th.  
 HIGH - 75th.  
 VHI, XTRAHI - 90th.

Select a rating for analyst capability (ACAP).

<u>VLOW</u>	<u>LOW</u>	NOM
HI	VHI	XTRAHI

Ratings AEXP: VLOW - <= 4 months experience.

LOW - 1 year.

NOM - 3 years.

HIGH - 6 years.

VHI, XTRAHI - 12 years.

Select a rating for applications experience (AEXP).

VLOW

LOW

NOM

HI

VHI

XTRAHI

Ratings PCAP: VLOW - 15th percentile.

LOW - 35th.

NOM - 55th.

HIGH - 75th.

VHI, XTRAHI - 90th.

Select a rating for programmer capability (PCAP).

VLOW

LOW

NOM

HI

VHI

XTRAHI

Ratings VEXP: VLOW - <= 1 month.

LOW - 4 months.

NOM - 1 year.

HIGH, VHI, XTRAHI - 3 years.

Select a rating for virtual machine experience (VEXP).

VLOW

LOW

NOM

HI

VHI

XTRAHI

Ratings LEXP: VLOW - 1 month experience.

LOW - 4 months.

NOM - 1 year.

HIGH, VHI, XTRAHI - 3 years.

Select a rating for programming language experience (LEXP).

VLOW

LOW

NOM

HI

VHI

XTRAHI

Ratings MODP: VLOW - no use.

LOW - beginning use.

NOM - some use.

HIGH - general use.

VHI, XTRAHI - routine use.

Select a rating for modern programming practices (MODP).

VLOW

LOW

NOM

HI

VHI

XTRAHI

Ratings TOOL: VLOW - basic microprocessor tools.  
LOW - basic mini tools.  
NOM - basic midi/maxi tools  
HIGH - strong maxi programming, test tools.  
VHI, XTRAHI - add requirements, design, management,  
documentation tools.

Select a rating for use of software tools (TOOL).

VLOW	LOW	<u>NOM</u>
HI	VHI	XTRAHI

Ratings SCED: VLOW - 75 % of nominal.

LOW - 85 %.  
NOM - 100 %.  
HIGH - 130 %.  
VHI, XTRAHI - 160 %.

Select a rating for required development schedule (SCED).

VLOW	LOW	<u>NOM</u>
HI	VHI	XTRAHI

---

You have chosen the DSS COCOMO model!

(Saving Data to Spreadsheet, Please Wait...)

VALUES SAVED!

---

(PRESS ANY KEY TO VIEW SPREADSHEET CALCULATIONS)

INTERMEDIATE COCOMO		ORGANIC MODE	EFFORT/SCHEDULE
KDSI =	5.00	COST DRIVERS	** CTRL F1 ** TO VIEW DEFINITIONS
MM =	17.34	RELY .75	** CTRL F2 ** TO VIEW COST DRIVER RATINGS
TDEV =	7.39	DATA .94	
EAF =	1.11	CPLX 1.00	** CTRL F3 ** TO SAVE RESULTS TO: DSSCOC.WKS AND EXIT SPREADSHEET
		TIME 1.00	
		STOR 1.00	** YOU MAY CHANGE KDSI AND COST DRIVER VALUES AS NEEDED
		VIRT .87	
		TURN 1.00	***** * For DSS projects * *****
		ACAP 1.19	
MMadj =	19.24	AEXP 1.29	
#PERS =	2.60	PCAP 1.00	
		VEXP 1.21	
		LEXP 1.07	
		MODP .91	
		TOOL 1.00	
		SCED 1.00	

Figure A.3

COST DRIVERS						
	VLOW	LOW	NOM	HIGH	VHI	XtraHI
RELY	.75	.88	1.00	1.15	1.40	1.40
DATA	.94	.94	1.00	1.08	1.16	1.16
CPLX	.70	.85	1.00	1.15	1.30	1.65
TIME	1.00	1.00	1.00	1.11	1.30	1.66
STOR	1.00	1.00	1.00	1.06	1.21	1.56
VIRT	.87	.87	1.00	1.15	1.30	1.30
TURN	.87	.87	1.00	1.07	1.15	1.15
ACAP	1.46	1.19	1.00	.86	.71	.71
AEXP	1.29	1.13	1.00	1.00	.91	.82
PCAP	1.42	1.17	1.00	.86	.70	.70
VEXP	1.21	1.10	1.00	.90	.90	.90
LEXP	1.14	1.07	1.00	.95	.95	.95
MODP	1.24	1.10	1.00	.91	.82	.82
TOOL	1.24	1.10	1.00	.91	.83	.83
SCED	1.23	1.08	1.00	1.04	1.10	1.10

\*\* (CTRL F5) TO RETURN TO MAIN SPREADSHEET

Figure A.4

FUNCTION POINT MODEL				EFFORT/SCHEDULE	
FUNCTION	Value	CPLX	COUNT	RATINGS	** CTRL F1 ** TO VIEW COMPLEXITY RATINGS
Inputs =	25	3	75	Q1. 1.00	
Outputs =	7	5	35	Q2. 3.00	
Inquiries =	5	3	15	Q3. 2.00	** CTRL F3 ** TO
Files =	6	7	42	Q4. 4.00	SAVE RESULTS TO:
Interfaces =	3	5	15	Q5. 2.00	DSSFP.WKS AND EXIT
Total =			182	Q6. 3.00	SPREADSHEET
				Q7. 2.00	
				Q8. 1.00	** YOU MAY CHANGE
RATINGS =	38.00			Q9. 4.00	FUNCTION VALUES
				Q10. 2.00	AND COMPLEXITY
FP =	187.46			Q11. 3.00	RATINGS AS NEEDED
				Q12. 4.00	
MM =	2.50			Q13. 2.00	
				Q14. 5.00	

Figure A.5

## APPENDIX B

### SOURCE CODE

The source code for the two knowledge bases (ICEM1.KBS) and (ICEM2.KBS) is displayed in sections A and B respectively.

#### A. (ICEM1.KBS) FILE

RUNTIME;

EXECUTE;

BKCOLOR=3;

ACTIONS

WOPEN 1,3,5,13,70,1

ACTIVE 1

color = 15

DISPLAY "

Welcome to the ICEM model!

The ICEM model is an integrated cost estimation model which uses an expert system to automate the Intermediate Constructive Cost Estimation Model (COCOMO), developed by Barry W. Boehm and the Function Point Model, developed by Allen Albrecht. Both of these models have been tailored to the unique environment of Decision Support Systems (DSS).

Press any key to continue!~"

WCLOSE 1

WOPEN 1,3,5,12,70,1

ACTIVE 1

color = 15

DISPLAY "

The ICEM model is actually three models in one. The COCOMO Intermediate model has been automated for all modes of operation including organic, semidetached and embedded modes. The second model specifically tailors the COCOMO semidetached mode for the DSS environment. The third model tailors the Function Point model for the DSS environment.

Press any key to begin the consultation.~"

WCLOSE 1

color = 0

CLS

FIND continue

review = YES

WHILETRUE model = COCOMO and review = YES THEN

CLS

WOPEN 1,3,5,12,70,1



```
ACTIVE 1
color = 15
DISPLAY "
```

The Intermediate COCOMO model uses 15 cost drivers applied to various attributes of a software project, the estimated number of thousand of delivered source instructions (KDSI), and the development mode to estimate the development time and cost of a software development project.

```
~"
CLS
DISPLAY "
```

The development mode of the project is determined by its size and complexity. The development mode may be considered either ORGANIC, SEMIDETACHED, or EMBEDDED. A listing of criteria for each mode follows. Determine the mode that best identifies your software project.

```
~"
CLS
DISPLAY "
1. ORGANIC
```

- a. Generally stable development environment.
- b. Minimal need for innovation in architectures of algorithms.
- c. Relatively small size.
- d. Relatively low premium on early completion of the project.
- e. Software project range usually not greater than 50 KDSI.
- f. Loose coupling with external systems.~"

```
CLS
DISPLAY "
2. SEMIDETACHED
```

- a. Mixture of organic and embedded characteristics.
- b. Intermediate level of experience with related systems.
- c. Wide mix of experienced and inexperienced people.
- d. Some experience with aspects of system under development.
- e. Software project range usually not greater than 300 KDSI.

```
~"
CLS
DISPLAY "
```

### 3. EMBEDDED

- a. Much innovation required.
- b. Integral part of some larger system with inflexibility.
- c. Interface requirements.
- d. High required reliability.
- e. Development within tight time and cost constraints.

~"

```
WCLOSE 1
color = 0
CLS
RESET review
FIND review
```

END

review = YES

WHILETRUE model = DSSCOC and review = YES THEN

```
CLS
WOPEN 1,1,5,20,70,1
ACTIVE 1
color = 15
DISPLAY "
```

This model is designed to be used as a cost estimation tool for Decision Support System projects that meet the requirements as specified by the Organic mode of the Intermediate COCOMO model. The following characteristics apply to projects which meet these requirements:

- a. Generally stable development environment.
- b. Minimal need for innovation in architectures of algorithms.
- c. Relatively small size.
- d. Relatively low premium on early completion of the project.
- e. Software project range usually not greater than 50 KDSI.
- f. Loose coupling with external systems.

~"

```
WCLOSE 1
WOPEN 1,3,5,12,70,1
ACTIVE 1
color = 15
CLS
DISPLAY "
```

This calibrated model estimates the development time and cost (in man-months) of a software project based on inputs of estimated number of thousand of delivered source instructions (KDSI), and values for 15 cost drivers.

Nominal values of 1.0 may be entered for unknown cost driver information.

```
~"
review = NO
WCLOSE 1
color = 0
END
review = YES
WHILETRUE model = DSSFP and review = YES THEN
WOPEN 1,3,5,12,70,1
ACTIVE 1
color = 15
CLS
DISPLAY "
```

The Function Point model makes effort estimates based on Allen Albrecht's function point sizing metric. metric based on five project factors which are measured by the user. These include:

1. Inputs
2. Outputs
3. Inquiries
4. Files
5. Interfaces

```
~"
WCLOSE 1
WOPEN 1,3,5,17,70,1
ACTIVE 1
color = 15
CLS
DISPLAY "
```

Due to difficulties in ex-post data gathering it was not possible to properly calibrate the model for the DSS environment. However, the framework for the model has been completed and the model has automated the process of determining the function point sizing metric. In order for the model to be able to make effort estimates on projects it must be calibrated to the users's environment. The method for calculating calibration coefficients is explained in Chapter V, section D of the thesis. Once the coefficients have been calculated they should be added to the (MM) effort equation located in cell B18 of the DSSFP.WKS spreadsheet.

Press any key to begin~"

```
review = NO
WCLOSE 1
color = 0
CLS
END
FIND finish;
```

!\*\*\*\*\* START RULES BLOCK \*\*\*\*\*

RULE 1

```
IF      retrieve = YES and file <> UNKOWN
THEN    Display "
        ABOUT TO VIEW SAVED SPREADSHEET FILE"
        COLOR = 30
        DISPLAY"
```

```
        (PRESS ENTER TO CONTINUE!)-"
        COLOR = 0
        SAVEFACTS fixvalue
        continue = NO;
```

RULE 2

```
IF      retrieve = NO and Model <> UNKOWN
THEN    continue = YES;
```

RULE 3

```
IF      continue=YES and model=COCOMO and mode=ORGANIC and
        kdsi=OK and LASTDRIVE=DONE
THEN    DISPLAY "
        You have chosen the {MODEL} {MODE} mode"
        COLOR = 30
        DISPLAY"
```

(SAVING DATA TO SPREADSHEET, PLEASE WAIT....)

"

```
PWKS kdsi_value, C4, c:\vpexp\organic
PWKS rrating, G6, c:\vpexp\organic
PWKS drating, G7, c:\vpexp\organic
PWKS crating, G8, c:\vpexp\organic
PWKS trating, G9, c:\vpexp\organic
PWKS srating, G10, c:\vpexp\organic
PWKS vrating, G11, c:\vpexp\organic
PWKS turrating, G12, c:\vpexp\organic
PWKS acrating, G13, c:\vpexp\organic
PWKS aerating, G14, c:\vpexp\organic
PWKS prating, G15, c:\vpexp\organic
PWKS verating, G16, c:\vpexp\organic
PWKS lrating, G17, c:\vpexp\organic
PWKS mrating, G18, c:\vpexp\organic
PWKS torating, G19, c:\vpexp\organic
PWKS scrating, G20, c:\vpexp\organic
```

```
RESET ALL
retrieve = NO
mode = ORGANIC
SAVEFACTS fixvalue
finish = DONE;
```

```

RULE 4
IF      continue=YES and model=COCOMO and mode=SEMIDETACHED and
        kdsi=OK and LASTDRIVE = DONE
THEN    DISPLAY "
        You have chosen the {MODEL} {MODE} mode"
        COLOR = 30
        DISPLAY"

```

```

        (SAVING DATA TO SPREADSHEET, PLEASE WAIT....)

```

```

"
PWKS kdsi_value, C4, c:\vpexp\semidet
PWKS rrating, G6, c:\vpexp\semidet
PWKS drating, G7, c:\vpexp\semidet
PWKS crating, G8, c:\vpexp\semidet
PWKS trating, G9, c:\vpexp\semidet
PWKS srating, G10, c:\vpexp\semidet
PWKS vrating, G11, c:\vpexp\semidet
PWKS turrating, G12, c:\vpexp\semidet
PWKS acrating, G13, c:\vpexp\semidet
PWKS aerating, G14, c:\vpexp\semidet
PWKS prating, G15, c:\vpexp\semidet
PWKS verating, G16, c:\vpexp\semidet
PWKS lrating, G17, c:\vpexp\semidet
PWKS mrating, G18, c:\vpexp\semidet
PWKS torating, G19, c:\vpexp\semidet
PWKS scrating, G20, c:\vpexp\semidet

```

```

RESET ALL
retrieve = NO
mode = SEMIDETACHED
SAVEFACTS fixvalue
finish = DONE;

```

```

RULE 5
IF      continue=Yes and model=COCOMO and mode=EMBEDDED and
        kdsi=OK and LASTDRIVE = DONE
THEN    DISPLAY "
        You have chosen the {MODEL} {MODE} mode"
        COLOR = 30
        DISPLAY"

```

```

        (SAVING DATA TO SPREADSHEET, PLEASE WAIT....)

```

```

"
PWKS kdsi_value, C4, c:\vpexp\organic
PWKS rrating, G6, c:\vpexp\organic
PWKS drating, G7, c:\vpexp\organic
PWKS crating, G8, c:\vpexp\organic
PWKS trating, G9, c:\vpexp\organic
PWKS srating, G10, c:\vpexp\organic

```

```

PWKS vrating, G11, c:\vpexp\organic
PWKS turrating, G12, c:\vpexp\organic
PWKS acrating, G13, c:\vpexp\organic
PWKS aerating, G14, c:\vpexp\organic
PWKS prating, G15, c:\vpexp\organic
PWKS verating, G16, c:\vpexp\organic
PWKS lrating, G17, c:\vpexp\organic
PWKS mrating, G18, c:\vpexp\organic
PWKS torating, G19, c:\vpexp\organic
PWKS scrating, G20, c:\vpexp\organic

```

```

RESET ALL
retreive = NO
mode = EMBEDDED
SAVEFACTS fixvalue
finish = DONE;

```

RULE 6

IF

```

continue = YES and model = DSSCOC and kdsi=OK and
LASTDRIVE = DONE

```

THEN

```

DISPLAY "
    You have chosen the DSS COCOMO model"
COLOR = 30
DISPLAY"

```

```

"          (SAVING DATA TO SPREADSHEET, PLEASE WAIT....)

```

```

PWKS kdsi_value, C4, c:\vpexp\dsscoc
PWKS rrating, G6, c:\vpexp\dsscoc
PWKS drating, G7, c:\vpexp\dsscoc
PWKS crating, G8, c:\vpexp\dsscoc
PWKS trating, G9, c:\vpexp\dsscoc
PWKS srating, G10, c:\vpexp\dsscoc
PWKS vrating, G11, c:\vpexp\dsscoc
PWKS turrating, G12, c:\vpexp\dsscoc
PWKS acrating, G13, c:\vpexp\dsscoc
PWKS aerating, G14, c:\vpexp\dsscoc
PWKS prating, G15, c:\vpexp\dsscoc
PWKS verating, G16, c:\vpexp\dsscoc
PWKS lrating, G17, c:\vpexp\dsscoc
PWKS mrating, G18, c:\vpexp\dsscoc
PWKS torating, G19, c:\vpexp\dsscoc
PWKS scrating, G20, c:\vpexp\dsscoc

```

```

RESET ALL
retreive = NO
model = dsscoc
SAVEFACTS fixvalue
finish = DONE;

```

RULE 7

```
IF          continue = YES and model = DSSFP
THEN        FIND inputs
            CLS
            DISPLAY "
```

Classifying inputs for complexity depends on two things: the number of files referenced or accessed, and the number of data items (fields or specific variables) referenced. Use the following chart for classifying inputs:

Press any key to view chart!

~"

CLS

DISPLAY "

	1-4 data items referenced	5-15	16 or more data items
0 or 1 file(s) referenced	Simple (3)	Simple (3)	Average (4)
2 files referenced	Simple (3)	Average (4)	Complex (6)
3 or more files ref.	Average (4)	Complex (6)	Complex (6)

"

FIND input\_cplx

CLS

FIND outputs

CLS

FIND output\_cplx

CLS

FIND inquiries

CLS

DISPLAY "

Classifying inquiries consists of two parts: classifying the inputs and classifying the outputs. The same charts are used for inputs and outputs as before. The only difference is that while a stand alone input actually updates the data store, an inquiry only directs the search and never updates. Use the following chart to classify the input inquiries:

Press any key to view chart!

~"

CLS

DISPLAY "

Input part:	1-4 data items referenced	5-15	16 or more data items
0 or 1 file(s) referenced	Simple (3)	Simple (3)	Average (4)
2 files referenced	Simple (3)	Average (4)	Complex (6)
3 or more files ref.	Average (4)	Complex (6)	Complex (6)

```

"
FIND in_inquir_cplx
CLS
FIND out_inquir_cplx
CLS
FIND inquir_cplx
CLS
FIND files
CLS
DISPLAY "

```

Classifying files is also a two step process which involves cross referencing the number of data items actually required by the application with the number of record formats within the file or the number of logical relationships in which the file participates are counted. Use the following chart to determine the file complexity factor:

Press any key to view chart!

```

~"
CLS
DISPLAY "

```

	1-19 data items ref.	20-50	51 or more data items
1 logical record format/relationship	Simple (7)	Simple (7)	Average (10)
2-5 logical record format/ relationships	Simple (7)	Average (10)	Complex (15)
6 or more logical record format/relationships	Ave. (10)	Complex (15)	Complex (15)

```

"
FIND file_cplx

```



```
CLS
FIND interfaces
CLS
DISPLAY "
```

The classification scheme for interfaces is similar to the one used for files. Use the following chart to determine the interface complexity factor:

Press any key to view chart!

```
~"
CLS
DISPLAY "
```

	1-19 data items ref.	20-50	51 or more data items
1 logical record format/relationship	Simple (7)	Simple (7)	Average (10)
2-5 logical record format/ relationships	Simple (7)	Average (10)	Complex (15)
6 or more logical record format/relationships	Ave. (10)	Complex (15)	Complex (15)

```
"
FIND inter_cplx
CLS
FIND Q1
CLS
FIND Q2
CLS
FIND Q3
CLS
FIND Q4
CLS
FIND Q5
CLS
FIND Q6
CLS
FIND Q7
CLS
FIND Q8
CLS
FIND Q9
CLS
FIND Q10
CLS
FIND Q11
```

```

CLS
FIND Q12
CLS
FIND Q13
CLS
FIND Q14
CLS

DISPLAY "
    You have chosen the DSS Function Point model"
COLOR = 30
DISPLAY"

```

(SAVING DATA TO SPREADSHEET, PLEASE WAIT....)

```

"
PWKS inputs, B6, c:\vpexp\dssfp
PWKS outputs, B7, c:\vpexp\dssfp
PWKS inquiries, B8, c:\vpexp\dssfp
PWKS files, B9, c:\vpexp\dssfp
PWKS interfaces, B10, c:\vpexp\dssfp
PWKS input_cplx, D6, c:\vpexp\dssfp
PWKS output_cplx, D7, c:\vpexp\dssfp
PWKS inquir_cplx, D8, c:\vpexp\dssfp
PWKS file_cplx, D9, c:\vpexp\dssfp
PWKS inter_cplx, D10, c:\vpexp\dssfp

PWKS Q1, H6, c:\vpexp\dssfp
PWKS Q2, H7, c:\vpexp\dssfp
PWKS Q3, H8, c:\vpexp\dssfp
PWKS Q4, H9, c:\vpexp\dssfp
PWKS Q5, H10, c:\vpexp\dssfp
PWKS Q6, H11, c:\vpexp\dssfp
PWKS Q7, H12, c:\vpexp\dssfp
PWKS Q8, H13, c:\vpexp\dssfp
PWKS Q9, H14, c:\vpexp\dssfp
PWKS Q10, H15, c:\vpexp\dssfp
PWKS Q11, H16, c:\vpexp\dssfp
PWKS Q12, H17, c:\vpexp\dssfp
PWKS Q13, H18, c:\vpexp\dssfp
PWKS Q14, H19, c:\vpexp\dssfp

```

```

RESET ALL
retrieive = NO
model = DSSFP
SAVEFACTS fixvalue
finish = DONE;

```

```

RULE 8
IF      out_inquir_cplx = 7
THEN    inquire_cplx = 7;

RULE 9
IF      out_inquir_cplx = 5 and in_inquir_cplx = 6
THEN    inquire_cplx = 6;

RULE 10
IF      out_inquir_cplx = 5 and in_inquir_cplx < 6
THEN    inquire_cplx = 5;

RULE 11
IF      out_inquir_cplx = 4 and in_inquir_cplx = 6
THEN    inquire_cplx = 6;

RULE 12
IF      out_inquir_cplx = 4 and in_inquir_cplx < 6
THEN    inquire_cplx = 4;

RULE 13
IF      kdsi_value <= 0
THEN    WHILE TRUE kdsi_value <= 0 THEN
        CLS
        COLOR = 30
        DISPLAY "
            You must enter a value greater than 0
            You have entered {kdsi_value} kdsi.
        "
        COLOR = 0
        RESET kdsi_value
        FIND kdsi_value
    END
    DISPLAY "
        You have entered {kdsi_value} kdsi."
    COLOR = 30
    DISPLAY "
        (Press Enter to continue)
    ~"
    COLOR = 0
    kdsi = OK;

RULE 14
IF      kdsi_value > 0
THEN    DISPLAY "
        You have entered {kdsi_value} kdsi."
    COLOR = 30
    DISPLAY "

```

(Press enter to continue)

```
~"
COLOR = 0
kdsi = OK;
```

RULE 15

```
IF      kdsi = ok
THEN    CLS
        WOPEN 2,3,5,16,70,1
        ACTIVE 2
        COLOR = 15
        DISPLAY"
```

Cost drivers are factors to consider in developing a model for estimating the cost of a software project. The drivers are grouped into four categories: software product attributes, computer attributes, personnel attributes, and project attributes.

Press enter to continue~"

```
CLS
DISPLAY"
```

The product attributes are:

RELY - required software reliability,  
DATA - data base size, and  
CPLX - product complexity.

Press enter to continue~"

```
CLS
DISPLAY"
```

The computer attributes are:

TIME - execution time constraint,  
STOR - main storage constraint,  
VIRT - virtual machine volatility, and  
TURN - computer turnaround time.

Press enter to continue~"

CLS  
DISPLAY"

The personnel attributes are:

ACAP - analyst capability,  
AEXP - applications experience,  
PCAP - programmer capability,  
VEXP - virtual machine experience, and  
LEXP - programming language experience.

Press enter to continue~"

CLS  
DISPLAY"

The project attributes are:

MODP - modern programming practices,  
TOOL - use of software tools  
SCED - required development schedule.

Press enter to continue~"

CLS  
DISPLAY "

Each of these cost driver attributes determines a multiplying factor which estimates the effect of the attribute on software development. These multipliers are applied to a nominal COCOMO development effort estimate to obtain a refined estimate of software development effort.

Press enter to continue~"

WCLOSE 2  
COLOR = 0  
CLS  
SHOW = DONE;

RULE 16  
IF SHOW = DONE and RELY = VLOW and driver2 = done

```

THEN      RRATING = .75
          lastdrive = DONE;

RULE 17
IF        SHOW = DONE and RELY = LOW and driver2 = done
THEN      RRATING = .88
          LASTDRIVE = DONE;

RULE 18
IF        SHOW = DONE and RELY = NOM and driver2 = done
THEN      RRATING = 1.00
          LASTDRIVE = DONE;

RULE 19
IF        SHOW = DONE and RELY = VHI and driver2 = done
THEN      RRATING = 1.40
          LASTDRIVE = DONE;

RULE 20
IF        SHOW = DONE and RELY = XTRAHI and driver2 = done
THEN      RRATING = 1.40
          LASTDRIVE = DONE;

RULE 21
IF        SHOW = DONE and DATA = VLOW AND DRIVER3 = DONE
THEN      DRATING = .94
          DRIVER2 = DONE;

RULE 22
IF        DATA = LOW AND DRIVER3 = DONE
THEN      DRATING = .94
          DRIVER2 = DONE;

RULE 23
IF        DATA = NOM AND DRIVER3 = DONE
THEN      DRATING = 1.00
          DRIVER2 = DONE;

RULE 24
IF        DATA = HI AND DRIVER3 = DONE
THEN      DRATING = 1.08
          DRIVER2 = DONE;

RULE 25
IF        DATA = VHI AND DRIVER3 = DONE
THEN      DRATING = 1.16
          DRIVER2 = DONE;

RULE 26
IF        DATA = XTRAHI AND DRIVER3 = DONE

```

```

THEN      DRATING = 1.16
          DRIVER2 = DONE;

RULE 27
IF        CPLX = VLOW AND DRIVER4 = DONE
THEN      CRATING = .70
          DRIVER3 = DONE;

RULE 28
IF        CPLX = LOW AND DRIVER4 = DONE
THEN      CRATING = .85
          DRIVER3 = DONE;

RULE 29
IF        CPLX = NOM AND DRIVER4 = DONE
THEN      CRATING = 1.00
          DRIVER3 = DONE;

RULE 30
IF        CPLX = HI AND DRIVER4 = DONE
THEN      CRATING = 1.15
          DRIVER3 = DONE;

RULE 31
IF        CPLX = VHI AND DRIVER4 = DONE
THEN      CRATING = 1.30
          DRIVER3 = DONE;

RULE 32
IF        CPLX = XTRAHI AND DRIVER4 = DONE
THEN      CRATING = 1.65
          DRIVER3 = DONE;

RULE 33
IF        TIME = VLOW AND DRIVER5 = DONE
THEN      TRATING = 1.00
          DRIVER4 = DONE;

RULE 34
IF        TIME = LOW AND DRIVER5 = DONE
THEN      TRATING = 1.00
          DRIVER4 = DONE;

RULE 35
IF        TIME = NOM AND DRIVER5 = DONE
THEN      TRATING = 1.00
          DRIVER4 = DONE;

RULE 36
IF        TIME = HI AND DRIVER5 = DONE

```

```

THEN      TRATING = 1.11
          DRIVER4 = DONE;

RULE 37
IF        TIME = VHI AND DRIVER5 = DONE
THEN      TRATING = 1.30
          DRIVER4 = DONE;

RULE 38
IF        TIME = XTRAHI AND DRIVER5 = DONE
THEN      TRATING = 1.66
          DRIVER4 = DONE;

RULE 39
IF        STOR = VLOW AND DRIVER6 = DONE
THEN      SRATING = 1.00
          DRIVER5 = DONE;

RULE 40
IF        STOR = LOW AND DRIVER6 = DONE
THEN      SRATING = 1.00
          DRIVER5 = DONE;

RULE 41
IF        STOR = NOM AND DRIVER6 = DONE
THEN      SRATING = 1.00
          DRIVER5 = DONE;

RULE 42
IF        STOR = HI AND DRIVER6 = DONE
THEN      SRATING = 1.06
          DRIVER5 = DONE;

RULE 43
IF        STOR = VHI AND DRIVER6 = DONE
THEN      SRATING = 1.21
          DRIVER5 = DONE;

RULE 44
IF        STOR = XTRAHI AND DRIVER6 = DONE
THEN      SRATING = 1.56
          DRIVER5 = DONE;

RULE 45
IF        VIRT = VLOW AND DRIVER7 = DONE
THEN      VRATING = .87
          DRIVER6 = DONE;

RULE 46
IF        VIRT = LOW AND DRIVER7 = DONE

```



```

THEN      VRATING = .87
          DRIVER6 = DONE;

RULE 47
IF        VIRT = NOM AND DRIVER7 = DONE
THEN      VRATING = 1.00
          DRIVER6 = DONE;

RULE 48
IF        VIRT = HI AND DRIVER7 = DONE
THEN      VRATING = 1.15
          DRIVER6 = DONE;

RULE 49
IF        VIRT = VHI AND DRIVER7 = DONE
THEN      VRATING = 1.30
          DRIVER6 = DONE;

RULE 50
IF        VIRT = XTRAHI AND DRIVER7 = DONE
THEN      VRATING = 1.30
          DRIVER6 = DONE;

RULE 51
IF        TURN = VLOW AND DRIVER8 = DONE
THEN      TURRATING = .87
          DRIVER7 = DONE;

RULE 52
IF        TURN = LOW AND DRIVER8 = DONE
THEN      TURRATING = .87
          DRIVER7 = DONE;

RULE 53
IF        TURN = NOM AND DRIVER8 = DONE
THEN      TURRATING = 1.00
          DRIVER7 = DONE;

RULE 54
IF        TURN = HI AND DRIVER8 = DONE
THEN      TURRATING = 1.07
          DRIVER7 = DONE;

RULE 55
IF        TURN = VHI AND DRIVER8 = DONE
THEN      TURRATING = 1.15
          DRIVER7 = DONE;

RULE 56
IF        TURN = XTRAHI AND DRIVER8 = DONE

```

```

THEN      TURRATING = 1.15
          DRIVER7 = DONE;

RULE 57
IF        ACAP = VLOW AND DRIVER9 = DONE
THEN      ACRATING = 1.46
          DRIVER8 = DONE;

RULE 58
IF        ACAP = LOW AND DRIVER9 = DONE
THEN      ACRATING = 1.19
          DRIVER8 = DONE;

RULE 59
IF        ACAP = NOM AND DRIVER9 = DONE
THEN      ACRATING = 1.00
          DRIVER8 = DONE;

RULE 60
IF        ACAP = HI AND DRIVER9 = DONE
THEN      ACRATING = .86
          DRIVER8 = DONE;

RULE 61
IF        ACAP = VHI AND DRIVER9 = DONE
THEN      ACRATING = .71
          DRIVER8 = DONE;

RULE 62
IF        ACAP = XTRAHI AND DRIVER9 = DONE
THEN      ACRATING = .71
          DRIVER8 = DONE;

RULE 63
IF        AEXP = VLOW AND DRIVER10 = DONE
THEN      AERATING = 1.29
          DRIVER9 = DONE;

RULE 64
IF        AEXP = LOW AND DRIVER10 = DONE
THEN      AERATING = 1.13
          DRIVER9 = DONE;

RULE 65
IF        AEXP = NOM AND DRIVER10 = DONE
THEN      AERATING = 1.00
          DRIVER9 = DONE;

RULE 66
IF        AEXP = HI AND DRIVER10 = DONE

```

```

THEN      AERATING = .91
          DRIVER9 = DONE;

RULE 67
IF        AEXP = VHI AND DRIVER10 = DONE
THEN      AERATING = .82
          DRIVER9 = DONE;

RULE 68
IF        AEXP = XTRAHI AND DRIVER10 = DONE
THEN      AERATING = .82
          DRIVER9 = DONE;

RULE 69
IF        PCAP = VLOW AND DRIVER11 = DONE
THEN      PRATING = 1.42
          DRIVER10 = DONE;

RULE 70
IF        PCAP = LOW AND DRIVER11 = DONE
THEN      PRATING = 1.17
          DRIVER10 = DONE;

RULE 71
IF        PCAP = NOM AND DRIVER11 = DONE
THEN      PRATING = 1.00
          DRIVER10 = DONE;

RULE 72
IF        PCAP = H1 AND DRIVER11 = DONE
THEN      PRATING = .86
          DRIVER10 = DONE;

RULE 73
IF        PCAP = VHI AND DRIVER11 = DONE
THEN      PRATING = .70
          DRIVER10 = DONE;

RULE 74
IF        PCAP = XTRAHI AND DRIVER11 = DONE
THEN      PRATING = .70
          DRIVER10 = DONE;

RULE 75
IF        VEXP = VLOW AND DRIVER12 = DONE
THEN      VERATING = 1.21
          DRIVER11 = DONE;

RULE 76
IF        VEXP = LOW AND DRIVER12 = DONE

```

```

THEN      VERATING = 1.10
          DRIVER11 = DONE;

RULE 77
IF        VEXP = NOM AND DRIVER12 = DONE
THEN      VERATING = 1.00
          DRIVER11 = DONE;

RULE 78
IF        VEXP = HI AND DRIVER12 = DONE
THEN      VERATING = .90
          DRIVER11 = DONE;

RULE 79
IF        VEXP = VHI AND DRIVER12 = DONE
THEN      VERATING = .90
          DRIVER11 = DONE;

RULE 80
IF        VEXP = XTRAHI AND DRIVER12 = DONE
THEN      VERATING = .90
          DRIVER11 = DONE;

RULE 81
IF        LEXP = VLOW AND DRIVER13 = DONE
THEN      LRATING = 1.14
          DRIVER12 = DONE;

RULE 82
IF        LEXP = LOW AND DRIVER13 = DONE
THEN      LRATING = 1.07
          DRIVER12 = DONE;

RULE 83
IF        LEXP = NOM AND DRIVER13 = DONE
THEN      LRATING = 1.00
          DRIVER12 = DONE;

RULE 84
IF        LEXP = HI AND DRIVER13 = DONE
THEN      LRATING = .95
          DRIVER12 = DONE;

RULE 85
IF        LEXP = VHI AND DRIVER13 = DONE
THEN      LRATING = .95
          DRIVER12 = DONE;

RULE 86
IF        LEXP = XTRAHI AND DRIVER13 = DONE

```

```

THEN      LRATING = .95
          DRIVER12 = DONE;

RULE 87
IF        MODP = VLOW AND DRIVER14 = DONE
THEN      MRATING = 1.24
          DRIVER13 = DONE;

RULE 88
IF        MODP = LOW AND DRIVER14 = DONE
THEN      MRATING = 1.10
          DRIVER13 = DONE;

RULE 89
IF        MODP = NOM AND DRIVER14 = DONE
THEN      MRATING = 1.00
          DRIVER13 = DONE;

RULE 90
IF        MODP = HI AND DRIVER14 = DONE
THEN      MRATING = .91
          DRIVER13 = DONE;

RULE 91
IF        MODP = VHI AND DRIVER14 = DONE
THEN      MRATING = .82
          DRIVER13 = DONE;

RULE 92
IF        MODP = XTRAHI AND DRIVER14 = DONE
THEN      MRATING = .82
          DRIVER13 = DONE;

RULE 93
IF        TOOL = VLOW AND DRIVER15 = DONE
THEN      TORATING = 1.24
          DRIVER14 = DONE;

RULE 94
IF        TOOL = LOW AND DRIVER15 = DONE
THEN      TORATING = 1.10
          DRIVER14 = DONE;

RULE 95
IF        TOOL = NOM AND DRIVER15 = DONE
THEN      TORATING = 1.00
          DRIVER14 = DONE;

RULE 96
IF        TOOL = HI AND DRIVER15 = DONE

```

```

THEN      TORATING = .91
          DRIVER14 = DONE;

RULE 97
IF        TOOL = VHI AND DRIVER15 = DONE
THEN      TORATING = .83
          DRIVER14 = DONE;

RULE 98
IF        TOOL = XTRAHI AND DRIVER15 = DONE
THEN      TORATING = .83
          DRIVER14 = DONE;

RULE 99
IF        SCED = VLOW
THEN      SCRATING = 1.23
          DRIVER15 = DONE;

RULE 100
IF        SCED = LOW
THEN      SCRATING = 1.08
          DRIVER15 = DONE;

RULE 101
IF        SCED = NOM
THEN      SCRATING = 1.00
          DRIVER15 = DONE;

RULE 102
IF        SCED = HI
THEN      SCRATING = 1.04
          DRIVER15 = DONE;

RULE 103
IF        SCED = VHI
THEN      SCRATING = 1.10
          DRIVER15 = DONE;

RULE 104
IF        SCED = XTRAHI
THEN      SCRATING = 1.10
          DRIVER15 = DONE;

RULE 105
IF        retrieve = YES
THEN      finish = done;

```

!\*\*\*\*\* START QUESTIONS BLOCK \*\*\*\*\*

ASK retrieve: "Would your like to retrieve a previously saved file?";

CHOICES retrieve: YES, NO;

ASK file: "Select the file to be retrieved:

ORGANIC = COCOMO Organic Mode File  
SEMIDETACHED = COCOMO Semidetached Mode File  
EMBEDDED = COCOMO Embedded Mode File  
DSSCOC = DSS COCOMO Semidetached Mode File  
DSSFP = DSS Function Point File  
";

CHOICES file: ORGANIC, SEMIDETACHED, EMBEDDED, DSSCOC, DSSFP;

ASK model: "Which of the following models would you like to select?

COCOMO = Regular COCOMO  
DSSCOC = COCOMO Tailored for DSS environment  
DSSFP = Function Point model tailored for DSS  
environment  
";

CHOICES model: COCOMO, DSSCOC, DSSFP;

ASK review: "Would you like to review the mode characteristics again?";

CHOICES review: YES, NO;

ASK mode: "Which of the following modes best identifies your software project?";

CHOICES mode: ORGANIC, SEMIDETACHED, EMBEDDED;

ASK kdsi\_value: "What is your best estimation on the number of thousands of delivered source instructions your program will have?  
";

ASK RELY: "

Ratings RELY: VLOW - effect, slight inconvenience.  
LOW - easily recoverable losses.  
NOM - moderate, recoverable losses.  
HIGH - high financial loss.  
VHI AND XTRAHI - risk to human life.

Select a rating for required software reliability (RELY).";

CHOICES RELY: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK DATA: "

Ratings DATA: VLOW and LOW - DB bytes/ prog. DSI < 10.  
NOM - 10 <= D/P <= 100.  
HIGH - 100 <= D/P <= 1000.  
XTRAHI - D/P >= 1000.

Select a rating for data base size (DATA).";

CHOICES DATA: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK CPLX: "

Ratings CPLX: VLOW - straightline code.

LOW - straightforward nesting of structured programming (SP).

NOM - mostly simple nesting.

HIGH - highly nested SP operators.

VHI - reentrant and recursive coding.

XTRAHI-microcode level control.

Select a rating for product complexity (CPLX).";

CHOICES CPLX: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK TIME: "

Ratings TIME: VLOW, LOW, NOM - 50 % use of available execution time.

HIGH - 70 %.

VHI - 85 %.

XTRAHI - 95 %.

Select a rating for execution time constraint (TIME).";

CHOICES TIME: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK STOR: "

Ratings STOR: VLOW, LOW, NOM- 50 % use of available storage.

HIGH - 70 %.

VHI - 85 %.

XTRAHI - 95 %.

Select a rating for main storage constraint (STOR).";

CHOICES STOR: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK VIRT: "

Ratings VIRT: VLOW, LOW - major change: 12 months, minor: 1 month.

NOM - major change: 6 months, minor: 2 weeks

HIGH - major: 2 months, minor: 1 week.

VHI, XTRAHI - major: 2 weeks, minor: 2 days.

Select a rating for virtual machine volatility (VIRT).";

CHOICES VIRT: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK TURN: "

Ratings TURN: VLOW, LOW - interactive.

NOM - average turnaround < 4 hours.

HIGH - 4-12 hours.

VHI, XTRAHI - >12 hours

Select a rating for computer turnaround time (TURN).";

CHOICES TURN: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK ACAP: "

Ratings ACAP: VLOW - 15th percentile.



LOW - 35th.  
NOM - 55th.  
HIGH - 75th.  
VHI, XTRAHI - 90th.

Select a rating for analyst capability (ACAP).";  
CHOICES ACAP: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK AEXP: "

Ratings AEXP: VLOW - <= 4 months experience.  
LOW - 1 year.  
NOM - 3 years.  
HIGH - 6 years.  
VHI, XTRAHI - 12 years.

Select a rating for applications experience (AEXP).";  
CHOICES AEXP: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK PCAP: "

Ratings PCAP: VLOW - 15th percentile.  
LOW - 35th.  
NOM - 55th.  
HIGH - 75th.  
VHI, XTRAHI - 90th.

Select a rating for programmer capability (PCAP).";  
CHOICES PCAP: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK VEXP: "

Ratings VEXP: VLOW - <= 1 month.  
LOW - 4 months.  
NOM - 1 year.  
HIGH, VHI, XTRAHI - 3 years.

Select a rating for virtual machine experience (VEXP).";  
CHOICES VEXP: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK LEXP: "

Ratings LEXP: VLOW - 1 month experience.  
LOW - 4 months.  
NOM - 1 year.  
HIGH, VHI, XTRAHI - 3 years.

Select a rating for programming language experience (LEXP).";  
CHOICES LEXP: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK MODP: "

Ratings MODP: VLOW - no use.  
LOW - beginning use.  
NOM - some use.

HIGH - general use.  
VHI, XTRAHI - routine use.

Select a rating for modern programming practices (MODP).";  
CHOICES MODP: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK TOOL: "

Ratings TOOL: VLOW - basic microprocessor tools.  
LOW - basic mini tools.  
NOM - basic midi/maxi tools  
HIGH - strong maxi programming, test tools.  
VHI, XTRAHI - add requirements, design, management,  
documentation tools.

Select a rating for use of software tools (TOOL).";  
CHOICES TOOL: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK SCED: "

Ratings SCED: VLOW - 75 % of nominal.  
LOW - 85 %.  
NOM - 100 %.  
HIGH - 130 %.  
VHI, XTRAHI - 160 %.

Select a rating for required development schedule (SCED).";  
CHOICES SCED: VLOW, LOW, NOM, HI, VHI, XTRAHI;

ASK inputs: "

In measuring inputs each unique user data or control  
input that is performed by the user within the application  
in order to add, delete or update something should be counted.

What is the number of inputs your program will have?  
";

ASK input\_cplx: "

Select the input complexity factor:  
";  
CHOICES input\_cplx: 3,4, 6;

ASK outputs: "

In measuring outputs, each unique user data or control  
output procedurally generated that leaves the application  
boundary should be counted. This includes reports, messages  
to the user, and outputs to other applications.

What is the number of outputs your program will have?  
";  
ASK output\_cplx: "

Outputs are classified in a similar format as inputs.  
Use the following chart for classifying outputs:

	1-5 data items referenced	6-19	20 or more data items
0 or 1 file(s) referenced	Simple (4)	Simple (4)	Average (5)
2-3 files	Simple (4)	Average (5)	Complex (7)
4 or more files ref.	Average (5)	Complex (7)	Complex (7)

Select the output complexity factor:

";

CHOICES output\_cplx: 4, 5, 7;

ASK inquiries: "

In measuring inquiries each unique input/output combination in which an on-line user-defined input causes and generates an immediate on-line output by the application should be counted. Many inquiries are simply requests for specific data from a database.

What is the number of inquiries your program will have?

";

ASK in\_inquir\_cplx: "

Select the input inquiries complexity factor:

";

CHOICES in\_inquir\_cplx: 3, 4, 6;

ASK out\_inquir\_cplx: "

Use the following chart to classify the output inquiries:

Output Part:	1-5 data items referenced	6-19	20 or more data items
0 or 1 file(s) referenced	Simple (4)	Simple (4)	Average (5)
2-3 files	Simple (4)	Average (5)	Complex (7)
4 or more files ref.	Average (5)	Complex (7)	Complex (7)

Select the output inquiries complexity factor:

";

CHOICES out\_inquir\_cplx: 4, 5, 7;

ASK files: "

Files are counted in the logical not physical sense. Therefore, you are not simply counting the number of physical files within your application. Rather, only files that contain data stored in logical groupings within the application are counted as files. These files perform data storage functions for the application.

What is the number of files your program will have?

";

ASK file\_cplx: "

Select the file complexity factor:

";

CHOICES file\_cplx: 7, 10, 15;

ASK interfaces: "

Interfaces involve using data stored by another application but used by the current application. In measuring interfaces count every logical file that is sent to, shared with, or received from another application.

What is the number of interfaces your program will have?

";

ASK inter\_cplx: "

Select the interface complexity factor:

";

CHOICES inter\_cplx: 4, 5, 7;

ASK Q1: "

Answer each question based on a scale of 0 to 5 where,

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q1. Does the system require reliable backup and recovery?

Select the appropriate rating scale:

";

CHOICES Q1: 0, 1, 2, 3, 4, 5;

ASK Q2: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q2. Are data communications required?

Select the appropriate rating scale:

";

CHOICES Q2: 0, 1, 2, 3, 4, 5;

ASK Q3: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q3. Are data communications required?

Select the appropriate rating scale:

";

CHOICES Q3: 0, 1, 2, 3, 4, 5;

ASK Q4: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q4. Is performance critical?

Select the appropriate rating scale:

";

CHOICES Q4: 0, 1, 2, 3, 4, 5;

ASK Q5: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q5. Will the system run in an existing, heavily utilized operational environment?

Select the appropriate rating scale:

";

CHOICES Q5: 0, 1, 2, 3, 4, 5;

ASK Q6: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q6. Does the system require on-line data entry?

Select the appropriate rating scale:

";

CHOICES Q6: 0, 1, 2, 3, 4, 5;

ASK Q7: "

0 = no influence

3 = average

1 = incidental

4 = significant

2 = moderate

5 = essential

Q7. Does the on-line data entry require the input transactions to be built over multiple screens or operations?

Select the appropriate rating scale:

";

CHOICES Q7: 0, 1, 2, 3, 4, 5;

ASK Q8: "

0 = no influence

3 = average

1 = incidental

4 = significant

2 = moderate

5 = essential

Q8. Are the master files updated on-line?

Select the appropriate rating scale:

";

CHOICES Q8: 0, 1, 2, 3, 4, 5;

ASK Q9: "

0 = no influence

3 = average

1 = incidental

4 = significant

2 = moderate

5 = essential

Q9. Are the inputs, outputs, files, or inquiries complex?

Select the appropriate rating scale:

";

CHOICES Q9: 0, 1, 2, 3, 4, 5;

ASK Q10: "

0 = no influence

3 = average

1 = incidental

4 = significant

2 = moderate

5 = essential

Q10. Is the internal processing complex?

Select the appropriate rating scale:

";

CHOICES Q10: 0, 1, 2, 3, 4, 5;

ASK Q11: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q11. Is the code designed to be reusable?

Select the appropriate rating scale:

";

CHOICES Q11: 0, 1, 2, 3, 4, 5;

ASK Q12: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q12. Are conversion and installation included in the design?

Select the appropriate rating scale:

";

CHOICES Q12: 0, 1, 2, 3, 4, 5;

ASK Q13: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q13. Is the system designed for multiple installations in different organizations?

Select the appropriate rating scale:

";

CHOICES Q13: 0, 1, 2, 3, 4, 5;

ASK Q14: "

0 = no influence	3 = average
1 = incidental	4 = significant
2 = moderate	5 = essential

Q14. Is the application designed to facilitate change and ease of use by the user?

Select the appropriate rating scale:

";

CHOICES Q14: 0, 1, 2, 3, 4, 5;

B. (ICEM2.KBS) FILE

EXECUTE;

RUNTIME;

BKCOLOR=3;

ACTIONS

LOADFACTS fixvalue

FIND spreadsheet;

RULE 1

IF retrieve = YES and file = ORGANIC

THEN DISPLAY "

(CALLING SPREADSHEET)"

WORKSHEET VPP

WORKON c:\vpexp, c:\vpexp\organic

continue = NO

spreadsheet = found;

RULE 2

IF retrieve = YES and file = SEMIDETACHED

THEN DISPLAY "

(CALLING SPREADSHEET)"

WORKSHEET VPP

WORKON c:\vpexp, c:\vpexp\semidet

continue = NO

spreadsheet = found;

RULE 3

IF retrieve = YES and file = EMBEDDED

THEN DISPLAY"

(CALLING SPREADSHEET)"

WORKSHEET VPP

WORKON c:\vpexp, c:\vpexp\embedded

continue = NO

spreadsheet = found;

RULE 4

IF retrieve = YES and file = DSSCOC

THEN DISPLAY"

(CALLING SPREADSHEET)"

WORKSHEET VPP

WORKON c:\vpexp, c:\vpexp\dsscoc

continue = NO

spreadsheet = found;

RULE 5

IF retrieve = YES and file = DSSFP

THEN DISPLAY"

(CALLING SPREADSHEET)"

WORKSHEET VPP



```
WORKON c:\vpexp, c:\vpexp\dssfp
continue = NO
spreadsheet = found;
```

```
RULE 6
IF
THEN
```

```
retreive = NO and mode = ORGANIC
DISPLAY "
```

VALUES SAVED!"

```
COLOR = 30
DISPLAY"
```

```
(PRESS ANY KEY TO VIEW SPREADSHEET CALCULATIONS)~"
WORKSHEET VPP
WORKON c:\vpexp, c:\vpexp\organic
spreadsheet = found;
```

```
RULE 7
IF
THEN
```

```
retreive = NO and mode = SEMIDETACHED
DISPLAY "
```

VALUES SAVED!"

```
COLOR = 30
DISPLAY"
```

```
(PRESS ANY KEY TO VIEW SPREADSHEET CALCULATIONS)~"
WORKSHEET VPP
WORKON c:\vpexp, c:\vpexp\semidet
spreadsheet = found;
```

```
RULE 8
IF
THEN
```

```
retreive = NO and mode = EMBEDDED
DISPLAY "
```

VALUES SAVED!"

```
COLOR = 30
DISPLAY"
```

```
(PRESS ANY KEY TO VIEW SPREADSHEET CALCULATIONS)~"
WORKSHEET VPP
```

WORKON c:\vpexp, c:\vpexp\embedded  
spreadsheet = found;

RULE 9  
IF  
THEN

retreive = NO and model = DSSCOC  
DISPLAY "

VALUES SAVED!"

COLOR = 30  
DISPLAY"

(PRESS ANY KEY TO VIEW SPREADSHEET CALCULATIONS)~"  
WORKSHEET VPP  
WORKON c:\vpexp, c:\vpexp\dsscoc  
spreadsheet = found;

RULE 10  
IF  
THEN

retreive = NO and model = DSSFP  
DISPLAY "

VALUES SAVED!"

COLOR = 30  
DISPLAY"

(PRESS ANY KEY TO VIEW SPREADSHEET CALCULATIONS)~"  
WORKSHEET VPP  
WORKON c:\vpexp, c:\vpexp\dssfp  
spreadsheet = found;

### LIST OF REFERENCES

1. Keen, P.G. and Morton, M.S., Decision Support Systems, An Organizational Perspective, Addison-Wesley, Reading, Massachusetts, 1978.
2. Basili, V.R., "Resource Models," in Models and Metrics for Software Managements and Engineering, IEEE Computer Society Press, pp. 4-9, 1980.
3. Londeix, B., Cost Estimation for Software Development, Addison-Wesley, 1987.
4. Thibodeau, R., "An Evaluation of Software Cost Estimating Models," Final Technical Report, RADC-TR-81-144, General Research Corporation, June 1981.
5. Kemerer, C.F., "An empirical validation of software cost models," Communications of the ACM, Vol. 30, No. 5, May 1987.
6. Mohanty, S. N. "Software cost estimation: present and future," Software Practice and Experience, Vol. 11 (1981).
7. Basili, V.R., Selby, R.W. Jr., and Phillips, T.Y., "Metric Analysis and Data Validation Across Fortran Projects," IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, November 1983.
8. Jones, T.C., "Measuring programming quality and productivity," IBM Systems Journal, Vol. 17, No. 1, 1978.
9. DeMarco, T., Controlling Software Projects, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
10. Conte, S.D., Dunsmore, H.E., Shen, V.Y., Software Engineering Metrics and Models, Benjamin/Cummings, 1986.
11. Halstead, M.H., Elements of Software Science, Elsevier North-Holland, Amsterdam, 1977.
12. Hamer, P.G., and Frewin, G.D., "M.H. Halstead's Software Science--A Critical Examination," Software Engineering, IEEE, 1982, pp. 197-206.

13. Navlakha, J., "Software Productivity Metrics: Some Candidates and their evaluation," 1986 National Computer Conference Proceedings, Vol. 55, AFIPS.
14. Li, H.F., and Cheung, W.K., "An Empirical Study of Software Metrics," IEEE Transactions on Software Engineering, Vol. SE-13, No. 6, June 1987.
15. Shen, V.Y., Conte, S.D., and Dunsmore, H.E., "Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support," IEEE Transactions on Software Engineering, Vol. SE-9, No. 2, March 1983.
16. Basili, V.R., Models and Metrics for Software Management and Engineering, IEEE Catalog No. EHO-167.7, Institute of Electrical and Electronics Engineers, 1980.
17. Albrecht, A.J., "Measuring Application Development Productivity", Proceedings of the Joint Share/Guide/IBM Application Development Symposium, pp. 83-92, October 1979.
18. Albrecht, A.J. and Gaffney (Jr.), J.E., "Software Function Source Lines of Code and Development Effort Prediction: A Software Science Validation," IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, pp. 639-647, November 1983.
19. Boehm, B.W., Software Engineering Economics, Englewood Cliffs, New Jersey: Prentice-Hall, 1981.
20. Pressman, R.S., Software Engineering: A Practitioner's Approach, 2nd edition, McGraw-Hill, 1987.
21. Boehm, B.W., "Software Engineering Economics," IEEE Transactions on Software Engineering, Vol. SE-10, No. 1, pp. 4-21, January 1984.
22. Bailey, E.K. and others, "A Descriptive Evaluation of Automated Software Cost-Estimation Models," IDA paper p-1979, contract MDA 903-84-C-0031, Institute for Defense Analysis, October 1986.
23. Nelson, E.A., Management Handbook for the Estimation of Computer Programming Costs, AD-A648750, Systems Development Corp., October 31, 1966.
24. Wolverton, R.W., "The Cost of Developing Large-Scale Software," IEEE Transactions on Computers, June 1974, pp. 615-636.

25. Putnam, L.H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions on Software Engineering, July 1978, pp. 345-361.
26. Putnam, L.H., and Fitzsimmons, A., "Estimating Software Costs," Datamation, September 1979, pp. 189-198, October 1979, pp. 171-178 and November 1979, pp. 137-140.
27. Putnam, L.H., Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers, IEEE Catalog No. EHO 165-1, Institute of Electrical and Electronics Engineers, 1980.
28. Wiener-Ehrlich, W.K., Hamrick, J.R. and Rupolo, V.F., "Modeling software behavior in terms of a formal lifecycle curve: implication for software maintenance," IEEE Transactions on Software Engineering, 10(4), pp. 376-383, 1984.
29. Herd, J.R., Postak, J.N., Russell, W.E., and Stewart, K.R., "Software Cost Estimation Study-Study Results," Final Technical Report, RADC-TR-77-220, AD-A042264, Vol. I (of two), Doty Associates, Inc., Rockville, Maryland, June 1977.
30. Freiman, F.R., and Park, R.E., "PRICE Software Model-Version 3: An Overview," Proceedings, IEEE-PINY Workshop on Quantitative Software Models, IEEE Catalog No. TH0067-9, October 1979, pp. 32-41.
31. Walston, C.E., and Felix, C.P., "A Method of Programming Measurement and Estimation," IBM Systems Journal, Vol. 16, No. 1, 1977, pp. 54-73.
32. Black, R.K.D., Curnow, R.P., Katz, R., and Gray, M.D., "BCS Software Production Data," Final Technical Report, RADC-TR-77-116, Boeing Computer Services, Inc., March 1977. NTIS No. AD-A039852.
33. Carriere, W.M., and Thibodeau, R., "Development of a Logistics Software Cost Estimating Technique for Foreign Military Sales," Report CR-3-839, General Research Corp., June 1979.
34. Bailey, J.W., and Basili, V.R., "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering, IEEE/ACM/NBS, March 1981, pp. 107-116.
35. Dircks, H.F., "SOFPCOST: Grumman's software cost eliminating model," in IEEE NAECON 1981, May 1981.

36. Tausworthe, R.C., "Deep Space Network Software Cost Estimation Model," Jet Propulsion Laboratory, Pasadena, CA, JPL Publication 81-7, April 1981.
37. Jensen, R.W., "An improved macrolevel software development resource estimation model," in Proceedings of the 5th ISPA Conference, April 1983, pp. 88-92.
38. Jensen, R.W., and Lucas, S., "Sensitivity analysis of the Jensen software model," in Proceedings of the 5th ISPA Conference, April 1983, pp. 384-389.
39. Rubin, H.A., "Macro-estimation of Software Development Parameters; The Estimacs System," Softfair Proceedings, IEEE, July 1983, pp. 110-118.
40. Rubin, H.A., "A Comparison of Cost Estimation Tools," Proceeding, 8th International Conference of Software Engineering, August 1985, pp. 174-180.
41. Jones, T.C., Programming Productivity, McGraw-Hill, 1986.
42. Gordon Group, Before You Leap, Users Guide, Gordon Group 1986.
43. BIS, BIS Estimator User Manual, Version 4.4, BIS Applied SystemsLtd, London, UK, 1987.
44. Dreger, J.B., Function Point Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.
45. Sprague, R.H.Jr., and Watson, H.J., Decision Support Systems, 2nd edition, Prentice Hall, 1989.
46. Paperback Software, VP-Expert Rule-Based Expert System Development Tool, 1989.
47. Paperback Software, VP-Planner Plus: Enhanced Spreadsheet, Database, Graphics, and Report Generation, 1987.
48. Verner, J., and Tate, G., "Estimating Size and Effort in Fourth-Generation Development," IEEE Software, July 1988, pp. 15-22.